

**Современный
Гуманитарный
Университет**

Дистанционное образование

0670.02.01;2

Рабочий учебник

Фамилия, имя, отчество _____

Факультет _____

Номер контракта _____

ДИСКРЕТНАЯ МАТЕМАТИКА

ЮНИТА 2

**ГРАФЫ И СЕТИ. КОДИРОВАНИЕ. АВТОМАТЫ И
АЛГОРИТМЫ**

МОСКВА 2001

Рекомендовано Министерством
образования Российской Федерации
в качестве учебного пособия для
студентов высших учебных заведений

КУРС: ДИСКРЕТНАЯ МАТЕМАТИКА

Юнита 1. Отношения. Булевы функции. Предикаты.

Юнита 2. Графы и сети. Кодирование. Автоматы и алгоритмы.

ЮНИТА 2

Содержатся основные понятия и факты, а также ряд приложений из разделов “Элементы комбинаторики”, “Теория графов и сетей”, “Теория кодирования”, “Конечные автоматы”, “Теория алгоритмов и вычислимых функций”. Изложение сопровождается большим числом примеров и упражнений.

Для студентов Современного Гуманитарного Университета

Юнита соответствует профессиональной образовательной программе №1

0670.005.02.01.01;2/11.13.Тир.7000

(С) СОВРЕМЕННЫЙ ГУМАНИТАРНЫЙ УНИВЕРСИТЕТ, 2001

ОГЛАВЛЕНИЕ

ДИДАКТИЧЕСКИЙ ПЛАН	4
ЛИТЕРАТУРА	6
ПЕРЕЧЕНЬ УМЕНИЙ	7
ТЕМАТИЧЕСКИЙ ОБЗОР	10
ГЛАВА I. ЭЛЕМЕНТЫ КОМБИНАТОРИКИ	10
§1. Основные комбинаторные конфигурации	10
§2. Формулы пересчета числа комбинаторных конфигураций	13
§3. Перечисление перестановок	22
§4. Приложения к теории вероятностей и теоретической физике	24
ГЛАВА II. ГРАФЫ И СЕТИ	27
§1. Ориентированные и неориентированные графы. Способы задания графов	27
§2. Цепи. Циклы. Связность	33
§3. Деревья	36
§4. Игра двух лиц с открытой информацией	40
§5. Эйлеровы (четные) графы. Цикломатическое число	45
§6. Двухполюсные сети. Потоки в сетях	49
§7. Кратчайшие пути и цепи	53
§8. Раскраска графов	58
ГЛАВА III. ЭЛЕМЕНТЫ ТЕОРИИ КОДИРОВАНИЯ	60
§1. Алфавитное кодирование	60
§2. Оптимальное кодирование	66
§3. Коды с обнаружением и исправлением ошибок	70
ГЛАВА IV. ЛОГИЧЕСКИЕ СЕТИ И АВТОМАТЫ	79
§1. Схемы из функциональных элементов для систем булевых функций	79
§2. Логические сети	86
§3. Конечные автоматы	90
§4. Периодические последовательности в автомате	97
ГЛАВА V. АЛГОРИТМЫ И ВЫЧИСЛИМОСТЬ	100
§1. Требования к алгоритмическим процедурам	100
§2. Рекурсивные функции	105
§3. Машины Тьюринга	117
§4. Понятие формальной грамматики	128
ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ	131
ТРЕНИНГ УМЕНИЙ	139
ГЛОССАРИЙ*	

* Глоссарий расположен в середине учебного пособия и предназначен для самостоятельного заучивания новых понятий.

ДИДАКТИЧЕСКИЙ ПЛАН

Комбинаторные конфигурации. Размещения, сочетания с повторениями и без повторений, перестановки. Комбинаторные правила суммы и произведения. Формулы пересчета числа комбинаторных конфигураций. Принцип включения и исключения. Треугольник Паскаля. Биномиальные и полиномиальные коэффициенты. Перечисление перестановок. Приложения к теории вероятностей и теоретической физике.

Ориентированные и неориентированные графы. Элементы графа: вершины, ребра, дуги. Способы задания графов. Матрица инцидентий, матрица соседства вершин. Геометрическая реализация графа. Степень вершины. Цепь и путь, цикл и контур. Связность графа. Деревья. Остов графа. Линейное пространство циклов графа. Базис циклов. Эйлеровы графы. Эйлеровы и гамильтоновы пути и циклы. Задача о коммивояжере. Дискретная игра двух лиц с открытой информацией. Дерево игры. Стратегия. Выигрышная и беспроигрышная стратегии.

Многополюсные и двухполюсные сети. Параллельно-последовательные сети. Поток в двухполюсной сети. Теорема Форда-Фалкерсона о максимальном потоке и ее комбинаторные приложения. Задача о назначениях. Кратчайший путь и кратчайшая цепь в сети. Задачи о раскраске графов. Хроматическое число графа.

Кодирование. Проблемы хранения и передачи сообщений. Алфавитное и побуквенное кодирование. Разделимые коды. Префиксное кодирование. Оптимальное кодирование. Коды Фано. Помехоустойчивое кодирование. Метрика Хэмминга. Типы ошибок передачи сообщений. Коды с обнаружением и исправлением ошибок. Коды Хэмминга для исправления одиночной ошибки замещения.

Схемы из функциональных элементов. Сумматор параллельного действия. Элемент задержки. Логические сети. Функционирование в дискретном времени. Сумматор последовательного действия. Конечный автомат. Автоматный оператор. Способы задания конечного автомата: таблица переходов, граф переходов, канонические уравнения. Эквивалентность автомата и логической сети. Периодическая последовательность в автомате. Примеры неавтоматных операторов.

Интуитивное понятие об алгоритмической процедуре. Детерминированность, результативность, дискретность, локальность алгоритма. Применимость. Массовая алгоритмическая проблема. Разрешимость проблемы. Примитивно-рекурсивные функции и предикаты. Оператор минимизации. Частично-рекурсивные и

общерекурсивные функции и предикаты. Тезис Черча. Универсальная общерекурсивная функция. Диагональный метод.

Машины Тьюринга (МТ). Программа МТ. Применимость МТ к конфигурации. Вычисление функций машинами Тьюринга. Правильно вычисляемые функции. Композиции МТ. Теорема Тьюринга о вычислимости рекурсивных функций. Универсальная машина Тьюринга. Тезис Тьюринга и его связь с тезисом Черча. Понятие о сложности алгоритма и вычисления.

ЛИТЕРАТУРА

Базовая

- * 1. Дискретная математика и математические вопросы кибернетики / Под общ. ред. С.В. Яблонского и О.Б. Лупанова, М., "Наука", 1974.
- * 2. Кузнецов О.П., Адельсон-Вельский Г.М. Дискретная математика для инженеров. 2-е издание, М., "Энергоатомиздат", 1988.

Дополнительная

- 3. Нефедов В.Н., Осипова В.А. Курс дискретной математики: Учеб. пособие для студентов специальности. "Прикладная математика". М., МАИ, 1992.
- 4. Леденева Т.М. Спец. главы математики. Дискретная математика: Учеб. пособие для специальности "Информационные системы". Воронеж, 1997.
- * 5. Насыров З.Х. Учебное пособие по курсу "Дискретная математика" Множества, отношения, алгоритмы. Для студентов специальности "Прикладная математика". М., Обнинский институт атомной энергетики, факультет кибернетики, 1997.
- * 6. Даугавет А.И., Иголинский В.Г. и др. Дискретная математика: Учеб. пособие. С-Пб, Гос. электротехн. унив., 1997.
- * 7. Эвнин А.Ю. Дискретная математика: конспект лекций для специальности "Прикладная математика". Челябинск, изд. Южно-Уральского гос. университета, 1998.
- * 8. Кофман А. Введение в прикладную комбинаторику. М., "Наука", 1975.
- * 9. Липский В. Комбинаторика для программистов. М., "Мир", 1988.
- * 10. Успенский В.А. Алгоритм. Математическая энциклопедия, т.1. М., 1977.

Примечание. Знаком (*) отмечены работы, на основе которых составлен тематический обзор.

ПЕРЕЧЕНЬ УМЕНИЙ

№ п/п	Умения	Алгоритмы
1	Построить различные представления графа, заданного множеством вершин и списком ребер.	1. Построить реализацию заданного ориентированного графа G. 2. Построить матрицу инцидентий графа G. 3. Построить матрицу соседства вершин графа G. 4. Построить матрицу соседства вершин соотнесенного неориентированного графа $\hat{G}$.
2	Построить остов и базис циклов неориентированного графа без петель. Определить цикломатическое число графа.	1. Построить реализацию заданного неориентированного графа G. 2. Найти цикломатическое число графа G. 3. При заданном упорядочении множества вершин последовательным присоединением подходящих ребер выбрать остов графа G. 4. Составить список хорд относительно построенного остова. 5. Поочередно присоединяя к остову по одной хорде, выделять элементарный цикл, состоящий из этой хорды и ребер единственной цепи, связывающей (в остове) концы хорды. 6. Выразить в построенном базисе заданный элементарный цикл.
3	Для списка сообщений с заданным распределением частот построить код Фано. Определить стоимость кода	1. Упорядочить список сообщений по убыванию частот. При равенстве частот порядок может быть любым. 2. Разделить группу событий на 2 части, различающиеся возможно меньше по суммарной частоте. 3. Установить первый символ кода 0 для сообщений первой группы, 1 – для второй группы. 4. Повторять процедуру п. 3 деления на две части в каждой из групп до тех пор, пока в группе больше одной строки, добавляя каждый раз к коду справа 0 для первой части подразбиения и 1 для второй части. 5. Определить стоимость построенного кода.

4	Для заданного сообщения X построить код Хэмминга. Внести единичную ошибку замещения и проверить декодирование искаженного сообщения по методу Хэмминга.	1. Подготовить строку для сообщения X', отводя места с номерами, равными 2^k, для проверочных символов, а остальные разряды - для информационных символов сообщения X'. 2. Разместить знаки сообщения X в информационных разрядах. 3. Построить таблицу с двоичными номерами разрядов и внести в информационные разряды знаки сообщения X. 4. Для каждого из проверочных разрядов с номером 2^k определить строки, формирующие проверочные знаки кода X'. 5. Вычислить значения проверочных разрядов. 6. Объединяя информационные и проверочные разряды, получить искомое закодированное сообщение X'. 7. Внести ошибку замещения в один из разрядов и произвести декодирование.
5	Построить автомат, реализующий оператор, заданный уравнением, выражающим зависимость выходного значения от входной последовательности.	1. Определить входной и выходной алфавиты. 2. Определить необходимое множество Q внутренних состояний автомата, представляющих различные ситуации, требующие различной реакции. Для каждого состояния $q \in Q$ определить выходные значения и назначить переходы в следующее состояние в зависимости от входного значения. 3. Построить таблицу размерности $w \times p$ (w - число входных символов, p - число внутренних состояний автомата). 4. Построить матрицу переходов автомата, занося в клетки таблицы, определяемые входным символом и состоянием автомата, символ следующего состояния и выходное значение. 5. Построить граф (диаграмму) переходов автомата с множеством вершин Q и дугами, соответствующими входному символу и направленными в следующее состояние, с указанием выходного значения.

6	Построить таблицу переходов и граф переходов для машины Тьюринга, заданной программой. Применить МТ к заданной начальной конфигурации. Определить заключительную конфигурацию.	1. Построить матрицу размерности $m \times n$ (m - число символов внешнего алфавита, n - число состояний МТ). Построить таблицу переходов, перенося в клетки, определяемые левыми частями команд, соответствующие им правые части команд. 2. Построить граф переходов МТ, используя умение 5, указывая дополнительно на дугах символ сдвига головки МТ согласно таблице переходов. 3. Составить протокол применения МТ к начальной конфигурации: пользуясь графом или таблицей переходов, последовательно преобразовывать возникающие конфигурации. 4. Достигнув заключительного состояния, определить результат работы МТ.
---	---	---

ГЛАВА 1. ЭЛЕМЕНТЫ КОМБИНАТОРИКИ

§1. Основные комбинаторные конфигурации

1. Среди основных задач комбинаторики выделяют следующие: пересчет, перечисление, классификация и оптимизация. Если требуется определить количество элементов, обладающих некоторым свойством или совокупностью свойств, то это задача пересчета. Если при этом необходимо указать список элементов, то это задача перечисления. Если пересчет приводит к слишком большим числам, то отказываются от соответствующего перечисления и только классифицируют элементы с помощью какого-нибудь соотношения, - и тогда это задача классификации. В некоторых задачах на множестве решений можно ввести функцию величины и относительно этой функции рассматривать задачу оптимизации: найти экстремум функции на определенном множестве объектов, либо указать все или некоторые объекты, для которых достигается экстремальное значение.

2. **Комбинаторная конфигурация** - это расположение конечного множества элементов, удовлетворяющее ряду специальных свойств. К основным комбинаторным конфигурациям относятся сочетания, размещения и перестановки. Ряд других конфигураций может быть сведен к ним.

Набор (множество или кортеж) элементов $(x_{i1}, x_{i2}, \dots, x_{ik})$, составленный из элементов множества $X = (x_1, x_2, \dots, x_n)$, называется **выборкой объема k из n элементов**, или **(n, k) -выборкой**. Выборки, различающиеся составом элементов, всегда считаются различными.

Выборка называется **упорядоченной**, если порядок элементов в ней задан (т.е. она представляет из себя кортеж). Две упорядоченные выборки, различающиеся только порядком следования элементов, считаются различными. Если порядок элементов в выборке несуществен, то выборка называется **неупорядоченной**.

Упорядоченная (n, k) -выборка, в которой элементы могут повторяться, называется **(n, k) -размещением с повторениями**, или **размещением с повторениями из n элементов по k** . Если

* Жирным шрифтом выделены новые понятия, которые необходимо усвоить. Знание этих понятий будет проверяться при тестировании.

элементы (n, k) -выборки попарно различны, то она называется **(n, k) -размещением без повторений**, или **размещением без повторений из n элементов по k** .

(n, n) -размещения без повторений называются **n -перестановками**, или **перестановками из n элементов**.

Неупорядоченные (n, k) -выборки называются **сочетаниями: с повторениями** или **без повторений**. Заметим, что (n, k) -сочетание без повторений - это k -элементное подмножество n -элементного множества.

Если элементы в (n, k) -выборке не могут повторяться, то, очевидно, выполнено неравенство $k \leq n$. Для выборки с повторениями возможно условие $k > n$.

Замечание. Когда мы говорим о выборках с повторениями, это не значит, что в ней обязательно есть повторяющиеся элементы; размещения (соответственно, сочетания) без повторений составляют подмножество множества размещений (соответственно, сочетаний) с повторениями.

Пример. Выборка (b, d, e) может рассматриваться как $(n, 3)$ -выборка с повторениями, так и без них при любом $n \geq 3$; выборка (b, d, e, d) - только как $(n, 4)$ -выборка с повторениями при $n \geq 3$.

3. В комбинаторике можно выделить два основных правила: правило суммы и правило произведения.

Пусть X - конечное множество из n элементов. Тогда говорят, что один объект из X можно выбрать n способами, и пишут $|X| = n$. Если X и

Y - непересекающиеся множества и $|X| = n, |Y| = m$, то $|X \cup Y| = m + n$. Свойство может быть распространено на большее число множеств.

Пусть $\{X_i\}$ - система попарно не пересекающихся множеств: $X_i \cap X_j = \emptyset$, если $i \neq j$. Тогда

$$\left| \bigcup_{i=1}^n X_i \right| = \sum_{i=1}^n |X_i|. \quad (1)$$

Это **правило суммы**, или **правило альтернатив**.

Если объект $x \in X$ может быть выбран n способами и после каждого из таких выборов объект $y \in Y$ может быть выбран m способами, то выбор упорядоченной пары (x, y) может быть осуществлен $m \cdot n$ способами. Это **правило произведения**. Оно также распространяется на большее число множеств. Для системы множеств $\{X_i\}$, $i = 1, 2, \dots, k$, где $|X_i| = m_i$,

выбор упорядоченной последовательности из k объектов $(x_1, x_2, \dots, x_k)$, $x_i \in X_i$, может быть осуществлен $m_1 \cdot m_2 \cdot \dots \cdot m_k$ способами.

Пример. Автомобильные номера имеют формат **AZZZZAA** (A - любая буква русского алфавита, кроме Ё, Й, Ъ, Ы, Ь, которые не употребляются в номерах; Z - произвольная цифра). Число различных возможных номеров равно произведению $28 \cdot 10 \cdot 10 \cdot 10 \cdot 10 \cdot 28 \cdot 28 = 219520000$.

Для пересекающихся множеств выполнены более сложные соотношения. Нетрудно убедиться, что для двух множеств X, Y число элементов, принадлежащих хотя бы одному из них, т.е. их объединению, равно

$$|X \cup Y| = |X| + |Y| - |X \cap Y| \quad (2)$$

(ср. формулу вероятности суммы двух событий в теории вероятностей).

Для трех множеств X, Y, Z число элементов, принадлежащих их объединению, равно

$$|X \cup Y \cup Z| = |X| + |Y| + |Z| - |X \cap Y| - |X \cap Z| - |Y \cap Z| + |X \cap Y \cap Z|. \quad (3)$$

Упражнение. Постройте диаграммы Венна для общего случая пересечения двух и трех множеств и проверьте соотношения (2) и (3). Можно воспользоваться рис. 2 и 3 из юниты №1.

Соотношениям (2) и (3) можно придать другую форму. Если рассмотреть множества X, Y, Z как подмножества универсального

множества U , то $|\overline{X \cup Y}| = |U| - |X \cup Y|$, откуда

$$|\overline{X \cup Y}| = |U| - (|X| + |Y|) + |X \cap Y|. \quad (4)$$

Аналогично:

$$|\overline{X \cup Y \cup Z}| = |U| - (|X| + |Y| + |Z|) + (|X \cap Y| + |X \cap Z| + |Y \cap Z|) - |X \cap Y \cap Z|. \quad (5)$$

Равенства (4) и (5) представляют частные случаи **принципа включения и исключения**, называемого также принципом решета.

Пусть рассматриваются n свойств $P(1), P(2), \dots, P(n)$ и N_i - число элементов некоторого N -элементного множества, обладающих свойством $P(i)$, N_{ij} - число элементов, обладающих свойствами $P(i)$ и $P(j)$, и вообще $N_{i_1 i_2 \dots i_k}$ - число элементов, обладающих свойствами

$(P_{i_1}), (P_{i_2}), \dots, (P_{i_s})$. Тогда число $N(0)$ элементов, не обладающих ни одним из этих свойств, задается знакопеременной алгебраической суммой:

$$N(0) = N - \sum_i N_i + \sum_{i < j} N_{ij} - \dots + (-1)^s \cdot \sum_{i_1 < i_2 < \dots < i_s} N_{i_1 i_2 \dots i_s} + (-1)^n \cdot N_{12 \dots n} \quad (6)$$

§2. Формулы пересчета числа комбинаторных конфигураций

1. Число (n, k) -размещений без повторений A_n^k может быть определено с помощью правила произведения. Первый из k элементов размещения может быть выбран n способами, второй - $(n - 1)$ способами, поскольку элемент, выбранный первым, не должен быть повторен; аналогично, для третьего (если $k > 2$) элемента остается $(n - 2)$ способов и т.д. Всего k элементов могут быть выбраны $A_n^k = n \cdot (n - 1) \cdot \dots \cdot (n - k + 1)$ способами: произведение k убывающих на 1 сомножителей, начиная с n . По-другому, используя обозначение $n! = 1 \cdot 2 \cdot \dots \cdot n$, можно записать:

$$A_n^k = \frac{n!}{(n - k)!}.$$

Примеры. $A_7^3 = 7! / 4! = 7 \cdot 6 \cdot 5 = 210$; $A_7^4 = 7! / 3! = 7 \cdot 6 \cdot 5 \cdot 4 = 840$;

$$A_7^5 = 7! / 2! = 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 = 2520.$$

Если $k = 1$, то $A_n^1 = n! / (n - 1)! = n$;

если $k = n - 1$, то $A_n^{n-1} = n! / 1! = n!$;

если $k = n$, то $A_n^n = n! / 0! = [\text{для единообразия формул удобно считать, что } 0! = 1] = n!$.

Равенство чисел A_n^{n-1} и A_n^n естественно: если выбраны $(n - 1)$ элементов из n , то единственный оставшийся элемент может быть выбран одним способом, и это не изменяет числа A_n^{n-1} по сравнению с A_n^n .

Таким образом, число n -перестановок P_n равно $n!$.

(n, k)-размещения с повторениями - это кортежи (по-другому, слова) длины k в n-элементном алфавите. Используя правило произведения, нетрудно показать, что число различных (n, k)-размещений с повторениями $\hat{A}_n^k = n^k$. Действительно, каждый из n членов упорядоченной последовательности может быть выбран n способами, независимо от других, а число их равно k.

Примеры. 1) Число слов длины k в алфавите {0, 1} равно 2^k ; таково же число всех подмножеств k-элементного множества, поскольку множество однозначно задается своей характеристической функцией. Число k-значных двоичных чисел равно 2^{k-1} , т.к. первая цифра должна равняться 1.

2) Число k-значных десятичных чисел (т.е. имеющих ровно k знаков) равно $9 \cdot 10^{k-1}$: их можно рассматривать как слова в алфавите {0, 1, 2, ..., 9}, причем первая цифра должна быть отлична от 0.

2. Числа (n, k)-сочетаний без повторений обозначаются символами

C_n^k или $\binom{n}{k}$ и называются также **биномиальными коэффициентами**,

поскольку совпадают с коэффициентами формулы бинома Ньютона для n-й степени двучлена $x + y$:

$$(x + y)^n = \sum_{k=0}^n C_n^k \cdot x^k \cdot y^{n-k}. \quad (7)$$

Действительно, коэффициент при члене $x^k \cdot y^{n-k}$ равен числу способов, которыми из n одинаковых сомножителей $(x + y) \cdot (x + y) \cdot \dots \cdot (x + y)$ можно выбрать k (в них выбирается x, а в остальных y - произведение выбранных так множителей равно как раз $x^k \cdot y^{n-k}$).

Для пересчета (n, k)-сочетаний без повторений удобно рассмотреть разбиение множества всех (n, k)-размещений без повторений на классы эквивалентности по отношению "иметь одинаковый состав элементов"; сочетания, отличающиеся составом элементов, попадают в различные классы. В каждом из классов содержатся упорядоченные конфигурации из одних и тех же k различных элементов, т.е. они представляют k-перестановки, и в каждом классе их число равно k!.

Отсюда число C_n^k (n, k)-сочетаний без повторений равно $A_n^k / k!$, т.е.

$$C_n^k = \frac{n!}{(n-k)! \cdot k!} = \frac{n \cdot (n-1) \cdot \dots \cdot (n-k+1)}{k!} = \frac{n \cdot (n-1) \cdot \dots \cdot (n-k+1)}{1 \cdot 2 \cdot \dots \cdot k}. \quad (8)$$

Для небольших значений k удобнее последнее выражение для C_n^k , поскольку в нем и в числителе, и в знаменателе - по k сомножителей: в числителе - отрезок натурального ряда, начиная с n , по убыванию; в знаменателе - отрезок натурального ряда $[1, k]$, начиная с 1, по возрастанию. В частности, $C_n^0 = 1$, $C_n^1 = n$, $C_n^2 = n \cdot (n-1)/2$. Удобно также считать, что $C_n^k = 0$, если $k > n$.

Примеры.

$$C_8^3 = \frac{8 \cdot 7 \cdot 6}{1 \cdot 2 \cdot 3} = 56; \quad C_{10}^4 = \frac{10 \cdot 9 \cdot 8 \cdot 7}{1 \cdot 2 \cdot 3 \cdot 4} = 210.$$

Приведем некоторые свойства биномиальных коэффициентов.

1) $C_n^k = C_n^{n-k}$. Это следует как из формулы (8), так и из того, что выборка k элементов из n однозначно определяет дополнение: выборку $(n - k)$ оставшихся элементов из n . Отсюда и из предыдущих равенств следуют

$$C_n^n = C_n^0 = 1, \quad C_n^{n-1} = C_n^1 = n.$$

Пример. $C_6^2 = \frac{6 \cdot 5}{1 \cdot 2} = 15, \quad C_6^4 = \frac{6 \cdot 5 \cdot (4 \cdot 3)}{1 \cdot 2 \cdot (3 \cdot 4)} = 15.$

В последнем равенстве в числителе и в знаменателе выделены скобками одинаковые сомножители.

2) $\sum_{k=0}^n C_n^k = 2^n$ - следует из подстановки $x = y = 1$ в формулу (7).

Примеры. $1 + 5 + 10 + 10 + 5 + 1 = 32 = 2^5.$
 $1 + 6 + 15 + 20 + 15 + 6 + 1 = 64 = 2^6.$

3) $\sum_{k=0}^n (-1)^k \cdot C_n^k = 0$ - следует из подстановки $x = 1, y = -1$ в формулу (7).

Пример. Для $n = 6$: $1 - 6 + 15 - 20 + 15 - 6 + 1 = 0.$

Используя свойство 3, покажем справедливость формулы (6) - принципа включения и исключения. Действительно, объект считается один раз в N , если не обладает никаким свойством, и совсем не учитывается во всех остальных слагаемых. Объект, имеющий $t > 0$

свойств, принимается за 1 в общем числе N , за (-1) в $-\sum_i N_i$, за C_t^2 в $\sum_{i < j} N_{ij}$ и вообще за $(-1)^s \cdot C_t^s$ в сумме $(-1)^s \cdot \sum_{i_1 < i_2 < \dots < i_s} N_{i_1 i_2 \dots i_s}$. Но для $t > 1$ из

свойства 3 имеем:

$$1 - t + C_t^2 + \dots + (-1)^s C_t^s + \dots = 0.$$

Таким образом, всякий объект, не обладающий никаким из свойств, вносит 1 в правую часть формулы (6), а объект, обладающий $t > 1$ свойствами, вносит 0. Это доказывает формулу (6).

4) $\sum_{k=0}^n C_n^k \cdot k = n \cdot 2^{n-1}$. Чтобы убедиться в справедливости равенства,

представим, что для каждого сочетания из n элементов $\{a_1, a_2, \dots, a_n\}$ выписаны на отдельную карточку символы $a_{i_1}, a_{i_2}, \dots, a_{i_k}$, соответствующие элементам этого (k -элементного) сочетания: всего карточек - 2^n (по свойству 2). Тогда суммарное число символов на всех карточках можно подсчитать двояко:

а) Суммарное число символов на C_n^k карточках, содержащих ровно

k символов, равно $k \cdot C_n^k$; в левой части рассматриваемого равенства – общее число символов на всех карточках для $k = 0, 1, \dots, n$.

б) Если зачеркнуть некоторый символ a_i на всех карточках, на которых он написан, то на них останутся всевозможные сочетания из остальных $(n - 1)$ элементов (их число - 2^{n-1}); значит символ a_i написан ровно на 2^{n-1} карточках. Общее число символов $n \cdot 2^{n-1}$ - в правой части равенства.

Примеры. Для $n = 5$: $0 \cdot 1 + 1 \cdot 5 + 2 \cdot 10 + 3 \cdot 10 + 4 \cdot 5 + 5 \cdot 1 = 80 = 5 \cdot 2^4$.

Для $n = 6$: $0 \cdot 1 + 1 \cdot 6 + 2 \cdot 15 + 3 \cdot 20 + 4 \cdot 15 + 5 \cdot 6 + 6 \cdot 1 = 192 = 6 \cdot 2^5$.

5) $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$. Равенство следует из такого рассуждения.

Зафиксируем некоторый элемент x n -элементного множества. Совокупность всех k -элементных подмножеств последнего (их число - левая часть равенства) разбивается на два непересекающихся класса:

мужчин и n женщин, выбирается делегация k человек. Это можно сделать C_{m+n}^k способами. Все k -элементные подмножества нашего $(m + n)$ -элементного множества можно классифицировать по числу мужчин в делегации. k -элементную делегацию, содержащую s мужчин, можно получить, выбирая сначала s мужчин одним из C_m^s способов, а затем $(k - s)$ женщин одним из C_n^{k-s} способов. По правилу произведения число таких делегаций равно $C_m^s \cdot C_n^{k-s}$, а по правилу суммы общее число k -элементных делегаций такое, как в правой части равенства.

Пример. $C_{10}^4 = C_{7+3}^4 = C_7^0 \cdot C_3^4 + C_7^1 \cdot C_3^3 + C_7^2 \cdot C_3^2 + C_7^3 \cdot C_3^1 + C_7^4 \cdot C_3^0 =$
 $[\text{учитывая, что } C_3^4 = 0] = 7 \cdot 1 + 21 \cdot 3 + 35 \cdot 3 + 35 \cdot 1 = 210.$

3. Чтобы получить число (n, k) -сочетаний с повторениями $\hat{C}_n^k$, поставим в соответствие каждому такому сочетанию (т.е. неупорядоченной k -выборке из n -элементного множества) кортеж в двухбуквенном алфавите $\{0, 1\}$ следующим образом. Составим сначала кортеж из n натуральных чисел $\{a_i\}$, так что a_i равно числу i -х элементов n -множества, входящих в данную k -выборку. Некоторые a_i могут быть равны. Сумма n компонент этого кортежа равна k . Далее составим кортеж из нулей и единиц, заменив каждое число a_i на a_i единиц, перемежая их нулями:

$$\underbrace{11\dots1}_{a_1} \underbrace{01\dots0}_{a_2} \dots \underbrace{10\dots0}_{a_i=0} 0\dots$$

Если некоторое $a_i = 0$, то между соответствующими нулями не будет ни одной единицы.

Говоря по-другому, кортеж из $(n - 1)$ нулей определяет n мест: слева от всего кортежа и справа от каждого из нулей. На место i вставим a_i единиц (некоторые a_i , как сказано выше, могут равняться 0).

Например, $(4, 7)$ -сочетанию $(1, 1, 3, 4, 4, 4, 4)$ соответствует кортеж $(2, 0, 1, 4)$, а ему, в свою очередь, - кортеж 1100101111 ; $(4, 7)$ -сочетанию $(1, 2, 2, 2, 3, 3, 3)$ соответствует сначала кортеж $(1, 3, 3, 0)$, а затем кортеж 1011101110 .

В результате получаем набор из $(n - 1)$ нулей и k единиц, т.е. вектор длины $(n - k + 1)$. Обратно, каждому такому вектору соответствует (n, k) -сочетание: серии идущих подряд единиц определяют числа a_i - их сумма равна k . Поэтому:

$$\hat{C}_n^k = C_{n+k-1}^k = C_{n+k-1}^{n-1} \quad (9)$$

Равенство (9) можно получить и другим способом. Пусть $(c_1, c_2, \dots, c_k)$ - (n, k) -сочетание с повторениями, причем $c_1 \leq c_2 \leq \dots \leq c_k$. Сопоставим ему выборку $(d_1, d_2, \dots, d_k)$, где $d_1 = c_1 + 0, d_2 = c_2 + 1, d_3 = c_3 + 2, \dots, d_k = c_k + k - 1$. Все числа d_i различны - поэтому выборка $(d_1, d_2, \dots, d_k)$ может рассматриваться как $(n+k-1, k)$ -сочетание без повторений. Обратно, каждому $(n+k-1, k)$ -сочетанию без повторений соответствует (n, k) -сочетание, возможно с повторениями, откуда следует равенство (9).

Пример. $\hat{C}_3^4 = C_6^4 = C_6^2 = 6 \cdot 5 / 2 = 15$. В таблице 1.1 перечислены все 15 $(3, 4)$ -сочетаний с повторениями из 3 элементов по 4 вместе с соответствующими им кортежами и соответствующими $(6, 4)$ -сочетаниями без повторений.

Таблица 1.1

1, 1, 1, 1	4, 0, 0	111100	1 2 3 4
1, 1, 1, 2	3, 1, 0	111010	1 2 3 5
1, 1, 1, 3	3, 0, 1	111001	1 2 3 6
1, 1, 2, 2	2, 2, 0	110110	1 2 4 5
1, 1, 2, 3	2, 1, 1	110101	1 2 4 6
1, 1, 3, 3	2, 0, 2	110011	1 2 5 6
1, 2, 2, 2	1, 3, 0	101110	1 3 4 5
1, 2, 2, 3	1, 2, 1	101101	1 3 4 6
1, 2, 3, 3	1, 1, 2	101011	1 3 5 6
1, 3, 3, 3	1, 0, 3	100111	1 4 5 6
2, 2, 2, 2	0, 4, 0	011110	2 3 4 5
2, 2, 2, 3	0, 3, 1	011101	2 3 4 6
2, 2, 3, 3	0, 2, 2	011011	2 3 5 6
2, 3, 3, 3	0, 1, 3	010111	2 4 5 6
3, 3, 3, 3	0, 0, 4	001111	3 4 5 6

4. Рассмотрим всевозможные размещения с повторениями $a_1 a_2 \dots a_n$ из n элементов с дополнительным условием: в каждом из них b_1 элементов первого рода, b_2 элементов второго рода и вообще b_i элементов i -го рода ($i = 1, 2, \dots, r$). Естественно, выполнено равенство

$b_1 + b_2 + \dots + b_r = n$. Заменим b_i элементов i -го рода различающимися элементами $b_{i1}, \dots, b_{ib_i}$ так, чтобы все элементы стали различными, и получим $n!$ перестановок. Каждое исходное размещение дает $b_1! \cdot b_2! \cdot \dots \cdot b_r!$ перестановок. Следовательно, число исходных размещений равно

$$\frac{n!}{b_1! \cdot b_2! \cdot \dots \cdot b_r!}, \quad \text{где } b_1 + b_2 + \dots + b_r = n.$$

Это число называется **полиномиальным коэффициентом**: оно равно коэффициенту при произведении $x_1^{b_1} x_2^{b_2} \dots x_r^{b_r}$ в разложении по степеням переменных полинома $(x_1 + x_2 + \dots + x_r)^n$. Биномиальные коэффициенты представляют частный случай при $r = 2$.

5. Еще один пример применения принципа включения и исключения дает задача о беспорядках (другое название - задача о встречах): сколько существует перестановок $a_1, a_2, \dots, a_n$ чисел $1, 2, \dots, n$ таких, что $a_i \neq i$ при любом $i = 1, 2, \dots, n$?

Обозначим это число через D_n и воспользуемся формулой (6). Здесь N элементов - это $n!$ перестановок $a_1, a_2, \dots, a_n$, а свойство $P(i)$ выражается равенством $a_i = i$ ($i = 1, 2, \dots, n$). Тогда $N_{i_1 i_2 \dots i_r} = (n - r)!$ - число перестановок, оставляющих на месте r определенных символов.

Далее, в $\sum N_{i_1 i_2 \dots i_r}$ имеется C_n^r слагаемых - по числу способов выбора $i_1, i_2, \dots, i_r$ из $1, 2, \dots, n$. Применяя (6), находим, что

$$N(0) = n! - n \cdot (n-1)! + C_n^2 \cdot (n-2)! - \dots + (-1)^r \cdot C_n^r \cdot (n-r)! + \dots + (-1)^n \cdot 1 = n! - n \cdot (n-1)! + n \cdot ((n-1)/2) \cdot (n-2)! - \dots + (-1)^r \cdot (n \cdot (n-1) \cdot \dots \cdot (n-r+1)/r!) \cdot (n-r)! + \dots + (-1)^n = n! \cdot (1 - 1/2! + 1/3! - \dots + (-1)^r/r! + \dots + (-1)^n/n!).$$

Сумма $1 - 1/2! + 1/3! - \dots + (-1)^r/r! + \dots + (-1)^n/n!$, заключенная в скобки - это частная сумма бесконечного ряда для числа e^{-1} . Поэтому $N(0)$, равное D_n , есть целое число, ближайшее к $n!/e$, т.е.

$$D_n/n! \approx e^{-1}. \quad (10)$$

Вот простая интерпретация формулы (10). Если n писем для различных адресатов случайным образом разложены в n конвертах с написанными адресами, то с вероятностью, близкой к $1/e \approx 0.368$, ни один из n адресатов не получит предназначенного ему письма. Может показаться парадоксальным, однако эта вероятность практически одинакова и для 10, и для 10000 писем.

6. Одна из классических задач комбинаторики - определить число

способов разместить некоторое множество объектов в каком-то количестве "ящиков" так, чтобы были выполнены заданные ограничения.

Если даны два множества X , Y , причем $|X| = n$, $|Y| = m$, то всякая функция $f: X \rightarrow Y$ задает такое размещение, если считать X объектами, а Y - ящиками и трактовать $f(x_i) = y_j$ как помещение объекта x_i в ящик y_j . Другая интерпретация - раскраска: Y - множество "цветов" и $f(x)$ - цвет объекта x . Тогда задачу можно сформулировать как определение количества раскрасок с соблюдением некоторых ограничений.

Заметим, что без потери общности можно всегда считать, что $X = \{1, 2, \dots, n\}$, а $Y = \{1, 2, \dots, m\}$. Если не накладывать никаких ограничений на размещения, то число всех функций $f: X \rightarrow Y$ равно m^n , ибо это просто (m, n) -размещения с повторениями.

Если каждый ящик содержит не более одного объекта, то мы имеем (m, n) -размещения без повторений; при этом $m \geq n$. Их число равно A_m^n : из m упорядоченных ящиков мы выбираем n и кладем в них по одному объекту. Если же $n > m$, то такое размещение, очевидно, невозможно. Здесь вступает в силу так называемый **принцип Дирихле**, или принцип ящиков - если в m ящиках находятся $(m + 1)$ или больше камней, то хотя бы в одном из ящиков больше одного камня.

Если мы размещаем n объектов по m ящикам так, чтобы каждый ящик содержал упорядоченную последовательность, а не просто неупорядоченное множество помещенных в него объектов, то будем называть такое размещение упорядоченным размещением n объектов по m ящикам. Их число - обозначим его $[m]^{(n)}$ - равно, полагая (для $n = 0$) $[m]^{(0)} = 1$:

$$[m]^{(n)} = m \cdot (m + 1) \cdot \dots \cdot (m + n - 1) = A_{m+n-1}^n.$$

Доказательство. Будем строить упорядоченные размещения, добавляя по очереди новые объекты. Первый объект можно разместить m способами, помещая его в любой из пока пустых ящиков. Второй - $(m + 1)$ способами, ибо его можно разместить в одном из $(m - 1)$ пустых ящиков или в ящике, содержащем первый объект: перед ним или после него. В общем случае, если в ящике находятся s объектов, то очередной объект можно поместить в этот ящик $(s + 1)$ способами. Поэтому, если перед размещением i -го объекта в k -ом ящике находятся r_k ($k = 1, 2, \dots$,

m ; $\sum_{k=1}^m r_k = i - 1$) объектов, то его можно разместить всего

$$(r_1 + 1) + \dots + (r_m + 1) = (r_1 + \dots + r_m) + m = (i - 1) + m$$

способами. Тем самым проведен индукционный шаг доказательства.

На рис. 1.2 представлены $[3]^{[2]} = 3 \cdot 4 = 12$ упорядоченных размещений двух элементов a, b в трех ящиках.

a	b	
a		b
b	a	
b		a
a b		
b a		
	a	b
	b	a
	a b	
	b a	
		a b
		b a

Рис. 1.2

§3. Перечисление перестановок

Рассмотрим некоторые алгоритмы порождения (перечисления) последовательности всех перестановок n элементов.

а) Лексикографическая (алфавитная). Последовательность определяется индуктивно: для каждой перестановки указан первый

элемент, далее - все следующие за ним.

1, (все перестановки из $\{2, \dots, n\}$ в алфавитном порядке)

2, (все перестановки из $\{1, 3, \dots, n\}$ в алфавитном порядке)

3, (все перестановки из $\{1, 2, 4, \dots, n\}$ в алфавитном порядке)

.....

k, (все перестановки из $\{1, 2, \dots, k-1, k+1, \dots, n\}$ в алфавитном порядке)

.....

n, (все перестановки из $\{1, 2, \dots, n-1\}$ в алфавитном порядке).

Пример. Для $n = 3$:

1, 2, 3

1, 3, 2

2, 1, 3

2, 3, 1

3, 1, 2

3, 2, 1

б) Конфигурации (перестановки) располагаются в такой последовательности, что каждая (начиная со второй) отличается от предыдущей тем, что в ней поменялись местами ровно два элемента (такая ситуация и вместе с тем операция такой замены называется **транспозицией**). Такая последовательность перестановок может оказаться полезной, если при решении какой-то (например, комбинаторной) задачи с каждой перестановкой связаны некоторые вычисления и существует возможность использования частичных результатов, полученных для предыдущей перестановки, если последовательные перестановки мало отличаются друг от друга.

Существуют различные алгоритмы построения таких последовательностей, удобные в тех или иных отношениях, в частности для программирования.

в) Частным случаем предыдущего можно считать последовательность, в которой разница между двумя соседними перестановками еще меньше: они различаются транспозицией двух соседних элементов. Идея такого алгоритма - индуктивное построение. Пусть уже построена обладающая этим свойством последовательность элементов $1, 2, \dots, (n-1)$. Тогда требуемую последовательность перестановок элементов $1, 2, \dots, n$ получим, вставляя n всеми возможными способами (их число равно n) в каждую из перестановок элементов $1, 2, \dots, (n-1)$ на различные места, передвигая n попеременно вперед и назад $(n-1)!$ раз. Ниже идея алгоритма проиллюстрирована для $n = 2, 3, 4$ и (начало последовательности) для $n = 5$; элемент

n выделен жирным шрифтом; справа в каждом столбце отмечены транспозиции элементов 1, 2,..., ($n - 1$):

$n = 2$	$n = 3$	$n = 4$	$n = 5$
1 2	1 2 3	1 2 3 4	1 2 3 4 5
2 1	1 3 2	1 2 4 3	1 2 3 5 4
	3 1 2 12	1 4 2 3	1 2 5 3 4
	3 2 1 21	4 1 2 3 23	1 5 2 3 4
	2 3 1	4 1 3 2 32	5 1 2 3 4 34
	2 1 3	1 4 3 2	5 1 2 4 3 43
		1 3 4 2	1 5 2 4 3
		1 3 2 4 13	1 2 5 4 3
		3 1 2. 4 31	1 2 4 5 3
		3 1 4 2	1 2 4 3 5 24
		3 4 1 2	1 4 2 3 5 42
		4 3 1 2 12	1 4 2 5 3
		4 3 2 1 21	1 4 5 2 3
		3 4 2 1	1 5 4 2 3
		3 2 4 1	5 1 4 2 3 14
		3 2 1. 4 32	5 4 1 2 3 41
		2 3 1 4 23	4 5 1 2 3
		2 3 4 1	4 1 5 2 3
		2 4 3 1	4 1 2 5 3
		4 2 3 1 31	4 1 2 3 5 23
		4 2 1 3 13	4 1 3 2 5 32
		2 4 1 3	4 1 3 5 2
		2 1 4 3	4 1 5 3 2
		2 1 3 4	4 5 1 3 2
			5 4 1 3 2 41
			5 1 4 3 2 14
			1 5 4 3 2
			

§4. Приложения к теории вероятностей и теоретической физике

Предположим, что в некоторой ситуации имеется n возможных взаимно исключающих исходов, которые мы обозначим через $x_1, x_2, \dots$,

x_n . Припишем исходу x_i некоторое число $p_i = p(x_i)$, где p_i - действительное число, $p_i > 0$, $p_1 + p_2 + \dots + p_n = 1$. Если некоторое событие E случается при одном из исходов $x_{i_1}, \dots, x_{i_m}$ и не происходит в других случаях, то

определим вероятность события E равенством $p(E) = p_{i_1} + \dots + p_{i_m}$.

Приписывание начальных вероятностей $p_1, \dots, p_n$ является оценкой относительного правдоподобия различных исходов.

Существует много практических ситуаций, в которых представляется разумным рассматривать n исходов как равновозможные. Тогда мы принимаем $p_1 = p_2 = \dots = p_n = 1/n$. В этом случае вероятность события E , происходящего только при m возможных исходах, равна $p(E) = m/n$. В такой ситуации вычисление $p(E)$ становится чисто комбинаторной задачей на подсчет m - числа возможных исходов, дающих событие E .

Пусть имеется N урн, пронумерованных от 1 до N . Разместим в урны произвольным образом n шаров, где $n < N$. Найдем вероятность того, что каждая из урн с номерами от 1 до n содержит точно по одному шару. Эта вероятность зависит от двух условий:

- 1) различимы ли шары или нет;
- 2) имеет ли место принцип исключения (или несовместности), который не позволяет положить второй шар в урну, уже содержащую один шар.

Если n шаров различимы и принцип исключения не имеет места, то существует N^n способов размещения n шаров в N урнах и $n!$ способов размещения их по одному в каждую из урн с номерами 1, ..., n . Из этих условий и определяется вероятность

$$p(E) = \frac{n!}{N^n}. \quad (*)$$

Если шары различимы и принцип исключения имеет место, то первый шар может быть помещен в любую из N урн, следующий - в любую из $(N - 1)$ урн, i -й - в любую из $(N - i + 1)$ урн; следовательно, число способов размещения n шаров в N урнах равно $A_N^n = N \cdot (N - 1) \cdot \dots \cdot (N - n + 1)$. Эти шары могут быть размещены $n!$ способами в урнах с номерами 1, ..., n , и тогда искомая вероятность равна

$$p(E) = \frac{n!}{A_N^n} = 1/C_N^n. \quad (**)$$

Если шары неразличимы и принцип исключения не имеет места, то задача сводится к подсчету числа неотрицательных целочисленных

решений уравнения $x_1 + x_2 + \dots + x_n = n$, где x_i - число шаров в i -й урне. Оно равно числу сочетаний с повторениями из N элементов по n , т.е.

C_{N+n-1}^n . Одно из решений: $x_1 = x_2 = \dots = x_n = 1$, $x_{n+1} = \dots = x_N = 0$. В этом случае искомая вероятность равна

$$p(E) = 1 / C_{N+n-1}^n. \quad (***)$$

С физической точки зрения "неразличимость" означает, что все сочетания равновозможны.

Если шары неразличимы и имеет место принцип исключения, то число способов размещения шаров есть не что иное, как число сочетаний без повторений из N элементов по n , т.е. C_N^n . Выбор первых n урн является единственно возможным и, следовательно, вероятность равна

$$p(E) = 1 / C_N^n, \quad (****)$$

т.е. то же, что в случае (**). Таким образом, если имеет место принцип исключения, то вероятность не зависит от различимости шаров.

В статистической физике рассматривается некоторая совокупность n частиц (это могут быть протоны, электроны, мезоны, нейтроны, нейтрино или фотоны); каждая из них может находиться в любом из N "состояний" (это могут быть энергетические уровни). Макроскопическое состояние этой системы из n частиц задается вектором $x = (x_1, x_2, \dots, x_N)$, где x_i - число частиц, находящихся в i -ом состоянии. Вероятность P любого отдельного макроскопического состояния зависит от того, различимы ли эти частицы и подчиняются ли они принципу исключения Паули, который гласит, что никакие две (неразличимые) частицы не могут находиться в одном и том же состоянии. Если рассматриваемые частицы различимы и не подчиняются принципу исключения, то вероятность P дается формулой (*), и тогда говорят, что частицы подчиняются классической статистике Максвелла-Больцмана. Если частицы неразличимы и не подчиняются принципу исключения, то вероятность P дается формулой (***), и тогда говорят, что частицы подчиняются статистике Бозе-Эйнштейна. Таковы фотоны и пи-мезоны.

Если частицы неразличимы и подчиняются принципу исключения, то вероятность P дается формулой (****), и при этом говорят, что они подчиняются статистике Ферми-Дирака. Таковы электроны, протоны, нейтроны. Случай (**) различимых частиц, подчиняющихся принципу исключения, в физике не встречается.

При достаточно высоких температурах, когда число состояний N

велико и различные макроскопические состояния почти равновозможны, статистики Ферми-Дирака и Бозе-Эйнштейна практически совпадают со статистикой Максвелла-Больцмана. При низких температурах низкие энергетические уровни возможны чаще, чем высокие, и тогда приведенные модели следует различать.

ГЛАВА II. ГРАФЫ И СЕТИ

§1. Ориентированные и неориентированные графы. Способы задания графов

1. Граф - это система некоторых объектов вместе с некоторыми парами этих объектов, изображающая отношения связи между ними. Графами удобно изображаются сети коммуникаций, дискретные многошаговые процессы, системы бинарных отношений, химические структурные формулы, различные схемы и диаграммы и др.

Неориентированный граф (соответственно **ориентированный граф**, или **орграф**) G - система $G = (V, E, \Gamma)$, состоящая из множества элементов $V = \{v\}$, называемых вершинами графа, множества элементов $E = \{e\}$, называемых ребрами, и отображения $\Gamma: E \rightarrow V^2$, ставящего в соответствие каждому элементу $e \in E$ неупорядоченную (соответственно упорядоченную) пару элементов $v_1, v_2 \in V$, называемых концами ребра e .

$V \cup E$ образует множество элементов графа; при этом предполагается, что $E \cap V = \emptyset$. Отображение Γ определяет отношение **инцидентности** ребра с каждым из своих концов. Для графа $G = (V, E, \Gamma)$ употребляется также более короткое обозначение $G = (V, E)$ без указания инцидентностей, которые определяются контекстом. По количеству элементов графы делятся на конечные и бесконечные. Здесь мы будем рассматривать, в основном, конечные графы, не оговаривая этого специально.

Если $\Gamma(e) = (v_1, v_2)$ - упорядоченная пара (т.е. $(v_1, v_2) \neq (v_2, v_1)$ при $v_1 \neq v_2$), то ребро e называется **ориентированной дугой**, *исходящей из вершины v_1 и входящей в вершину v_2* ; v_1 называется началом, а v_2 - концом дуги e . Если $\Gamma(e) = (v_1, v_2)$ - неупорядоченная пара, то ребро e называется **неориентированным**. Всякому графу G можно поставить

в соответствие *соотнесенный неориентированный граф* $\hat{G}$ с теми же множествами V и E и инцидентностями, но все пары неупорядоченные.

Вершина, не инцидентная ни одному ребру, называется **изолированной**. Вершина, инцидентная ровно одному ребру, и само это ребро называются **концевыми**, или **висячими**. Ребро с совпадающими концами называется **петлей**. Две вершины, инцидентные одному и тому же ребру, называются соседними (или **смежными**). Два ребра, инцидентные одной и той же вершине, называются **смежными**. Ребра, которым поставлена в соответствие одна и та же пара вершин, называются **кратными**, или **параллельными**.

Для различных задач одному и тому же объекту можно сопоставить граф по-разному. Так, фрагмент дорожной сети может быть представлен неориентированным ребром, изображающим наличие связи между его концами (населенными пунктами, городскими площадями или перекрестками); одной или двумя дугами, передающими одностороннее или двустороннее движение транспорта; несколькими дугами - отдельно для каждого ряда движения. Ребрам могут быть приписаны также числа, означающие длину, рядность, уклон и другие численные или иные характеристики.

2. Часто бывает важно определить, какие графы считаются различными, а какие не различаются. Обычно это связывают с понятием изоморфизма графов. Два графа $G_1 = (V_1, E_1, \Gamma_1)$ и $G_2 = (V_2, E_2, \Gamma_2)$ называются изоморфными, если существуют взаимно однозначные отображения $f: V_1 \rightarrow V_2$ и $g: E_1 \rightarrow E_2$, сохраняющие инцидентность, т.е. такие, что для всякого $e \in E_1$ равенство $\Gamma_1(e) = (v_1, v_2)$ влечет за собой равенство $\Gamma_2(g(e)) = (f(v_1), f(v_2))$.

Во многих случаях можно рассматривать графы с точностью до изоморфизма, т.е. не различать изоморфные графы; однако, если какие-то вершины или ребра графов обладают различной индивидуальностью, например, они занумерованы или им сопоставлены какие-либо численные характеристики (вес ребра, длина ребра и др.), то естественно при сравнении двух графов эту индивидуальность учитывать.

3. Существует несколько способов задания графов, связанных с различной формой задания функции Γ . Вот некоторые из них для конечных графов.

1) Перечисление (список) ребер графа с указанием их концов и добавлением списка изолированных вершин.

2) **Матрица инцидентностей графа с b вершинами и p ребрами** - прямоугольная матрица $A = ||a_{ij}||$ с b строками и p столбцами, строки которой соответствуют вершинам графа, а столбцы - ребрам, причем

для неориентированного графа элемент матрицы a_{ij} равен 1, если вершина v_i и ребро e_j инцидентны, и равен 0 в противном случае. Для ориентированного графа $a_{ij} = -1$, если v_i является началом дуги e_j , и $a_{ij} = +1$, если v_i - конец дуги e_j . В каждом столбце матрицы инцидентностей - два ненулевых элемента, если ребро - не петля. Петле соответствует элемент, равный 2.

3) **Матрица соседства (смежности) вершин графа с b вершинами** - квадратная матрица $B = ||b_{ij}||$ размерности b , строки и столбцы которой соответствуют вершинам графа, причем неотрицательный элемент b_{ij} равен числу ребер, идущих из вершины v_i в вершину v_j (b_{ij} не равно, вообще говоря, b_{ji} , однако для неориентированных графов матрица соседства - симметричная). Для несмежных вершин соответствующий элемент матрицы равен 0.

Если матрица инцидентностей задает граф однозначно, то матрица соседства вершин определяет граф с точностью до замены любого неориентированного ребра парой противоположно направленных дуг между теми же вершинами. Однако для графов без кратных ребер задание графа и этой матрицей однозначно, элементы матрицы соседства равны в этом случае 0 или 1.

4) Для наглядности граф часто представляют в виде геометрического объекта: вершинам соответствуют различные выделенные точки в пространстве (на плоскости), ребрам - отрезки кривых, связывающие соответствующие точки и не проходящие через выделенные точки, отличные от их концов. Отношению инцидентности вершин и ребер графа соответствует при этом геометрическая инцидентность выделенных точек и линий. Кроме того, предполагается, что кривые попарно не пересекаются во внутренних точках. Такое представление графа называется **реализацией**.

Можно показать, что всякий граф с конечным (и даже счетным) числом элементов может быть реализован в трехмерном пространстве, причем, если граф не содержит кратных ребер, то ребра можно реализовать прямолинейными отрезками. На плоскости реализуется не всякий граф, что порождает деление графов на **плоские** и **неплоские**. Тем не менее даже неплоский граф бывает удобно изображать на плоскости, надо только отличать вершины от пересечений ребер на рисунке (например, изображать вершины кружками); ориентацию ребер показывают стрелками.

Если представить графом уличную (дорожную) сеть, где ребра изображают отрезки дорог, связывающие соседние вершины (площади и перекрестки), то для небольшого населенного пункта такой граф может быть плоским, но для города с путепроводами, мостами и

транспортными развязками в разных уровнях, он скорее всего неплоский.

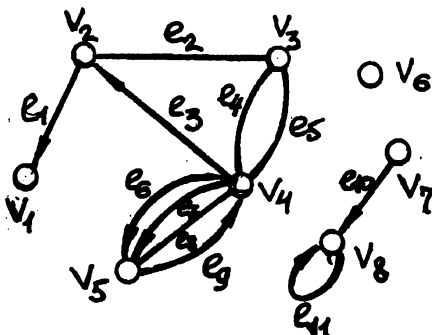


Рис. 2.1

На рис. 2.1 изображен граф с 8 вершинами и 11 ребрами и проиллюстрированы некоторые введенные понятия: ребра e_1 , e_3 , e_6 , e_7 , e_9 , e_{10} являются дугами; v_6 - изолированная вершина; e_4 и e_5 - параллельные ребра; e_6 , e_7 , e_8 , e_9 - также параллельные ребра, e_{11} - петля; v_2 и v_3 , v_2 и v_4 - пары соседних вершин; e_3 и e_4 - пара смежных ребер. Матрица инцидентий A и матрица соседства вершин B этого графа таковы:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & -1 & -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 12 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

4. Рассмотрим несколько примеров графов, имеющих интересные применения.

(1) **Полным** графом называется граф без кратных ребер (и иногда

без петель), в котором любые две вершины соединены ребром (ориентированным или неориентированным). Полный неориентированный граф с b вершинами обозначается K_b . Очевидно, граф K_b содержит $C_b^2 = b \cdot (b - 1) / 2$ ребер. Полный ориентированный граф иногда называют турниром, поскольку он отражает результат состязания по круговой системе в один круг. Полный граф представляет бинарное отношение R : если R рефлексивно, то граф - с петлями; если R симметрично, то граф - неориентированный, если R антисимметрично, то - ориентированный.

На рис. 2.2 а, б, в изображены графы K_3 , K_4 , K_5 . На рис. 2,2 г граф K_5 представлен с минимальным числом пересечений изображений ребер: устранить их полностью нельзя.

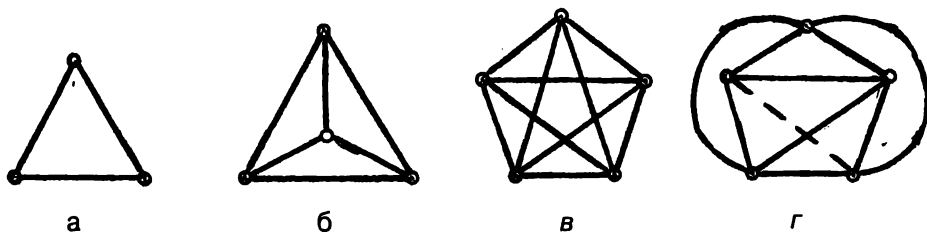


Рис. 2.2

(2) **Двудольным** графом называется граф, вершины которого разбиты на два непересекающихся класса: $V = V_1 \cup V_2$, а ребра связывают вершины только из разных классов - не обязательно все пары (рис. 2.3). Если же каждая из вершин класса V_1 связана ребром с каждой вершиной класса V_2 , то граф называется **полным двудольным** и обозначается $K_{m,n}$, где $m = |V_1|$, $n = |V_2|$. Очевидно, граф $K_{m,n}$ содержит $(m+n)$ вершин и $m \cdot n$ ребер. На рис. 2.3а изображен двудольный граф, на рис. 2.3б, в - полные двудольные графы $K_{3,2}$ и $K_{3,3}$ (последний носит название "3 дома - 3 колодца").

Упражнение. Изобразите граф $K_{3,3}$ с минимально возможным числом пересечений.

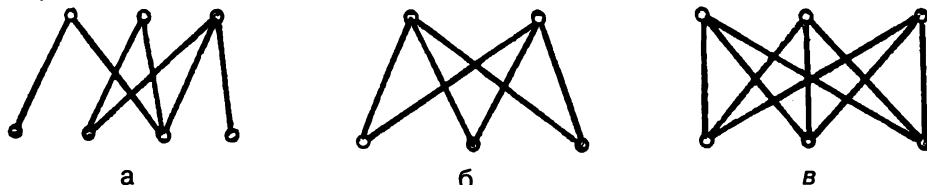


Рис. 2.3

(3) **n -мерным единичным кубом** называется граф E^n , вершинами которого являются все наборы длины n из нулей и единиц, а ребра соединяют вершины, различающиеся ровно в одной компоненте. Случаи $n = 3$ и $n = 4$ представлены на рис. 2.4 а, б.

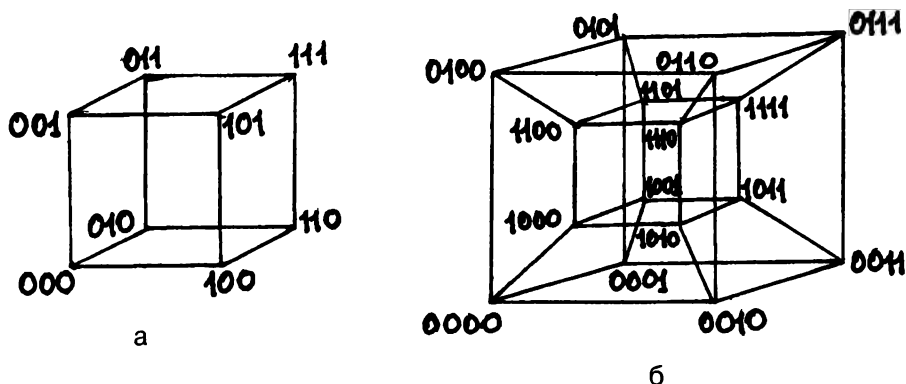


Рис. 2.4

Упражнение. Докажите, что графы E^n являются двудольными (указание: рассмотрите наборы с различным числом единиц).

Представление булевых функций как подмножеств вершин графа E^n рассматривалось в разделе “Булевы функции” юниты №1.

5. Граф $H = (V', E')$ называется подграфом графа $G = (V, E)$, обозначение: $H \subseteq G$, если $V' \subseteq V$, $E' \subseteq E$ и для множеств V' и E' сохраняются инциденции графа G . При этом, очевидно, каждое ребро из E' входит в подграф H вместе со своими концами.

Иногда рассматриваются только подграфы без изолированных вершин или только подграфы, содержащие все вершины графа (и только часть ребер); такие подграфы полностью определяются множеством своих ребер. В этих случаях можно естественным образом определить теоретико-множественные операции над подграфами: пересечение, объединение, симметрическую разность (называемую также **суммой по модулю 2** или просто суммой), дополнение до всего графа. **Подграфом, натянутым на множество вершин $V' \subseteq V$ графа G ,** называется подграф, содержащий вершины из V' и все ребра графа G , соединяющие пары вершин из V' .

Подграф Z_α , состоящий из всех ребер, инцидентных вершине α , называется **звездой вершины α** . Для графов без петель **степень $s(\alpha)$**

вершины α есть число ребер в звезде $Z\alpha$. Очевидно, что сумма степеней всех вершин графа без петель равна удвоенному числу ребер.

Для графа, изображенного на рис. 1, $s(v_1) = 1$, $s(v_2) = s(v_3) = 3$, $s(v_4) = 7$, $s(v_6) = 0$.

Если все вершины графа имеют одинаковую степень s (такой граф называют однородным), то сумма степеней его вершин равна $b \cdot s$, а число ребер равно $b \cdot s / 2$.

Упражнение. Определите число ребер в графах E^3 , E^4 (рис. 4а,б).

§2. Цепи. Циклы. Связность.

1. Последовательность вершин и ребер графа G

$$v_{i_0} e_1 v_{i_2} e_2 v_{i_2} \dots v_{i_{n-1}} e_n v_{i_n} \quad (*)$$

называется **путем** $[v_{i_0}, v_{i_n}]$ **из вершины** v_{i_0} **в вершину** v_{i_n} , если

$\Gamma(e_k) = (v_{i_{k-1}}, v_{i_k})$ для $k = 1, 2, \dots, n$. Вершина v_{i_0} называется **началом**, а

v_{i_n} - **концом пути**; число n называется **длиной пути** (путь нулевой длины состоит из одной вершины). В общем случае среди вершин и ребер последовательности $(*)$ могут быть повторения. Каждое ребро пути может быть ориентированным (обязательно из $v_{i_{k-1}}$ в v_{i_k}) или

неориентированным. Говорят, что путь $[v_{i_0}, v_{i_n}]$ проходит

последовательно через вершины $v_{i_0}, v_{i_1}, \dots, v_{i_n}$ по ребрам $e_1, e_2, \dots, e_n$.

Путь естественно рассматривать как непрерывную траекторию движения по вершинам и ребрам графа. Для графа без кратных ребер путь можно задавать последовательностью вершин.

Цепью называется последовательность вершин и ребер, образующая путь в соотнесенном неориентированном графе. Отсюда всякий путь является цепью, обратное верно только для неориентированных графов, в которых понятия пути и цепи совпадают.

Если последовательность $(*)$ - цепь, то, очевидно, и ее обращение $v_{i_n} e_n$

$v_{i_{n-1}} e_{n-1} v_{i_{n-2}} \dots v_{i_1} e_1 v_{i_0}$ (т.е. запись в обратном порядке) - цепь. В дальнейшем эти цепи различать мы не будем и будем говорить о цепи между вершинами v_{i_0} и v_{i_n} .

Если в последовательности (*) $v_{i_0} = v_{i_n}$ ($n > 0$), т.е. начальная вершина совпадает с конечной, то такой путь (соответственно цепь) называется **контуром** (соответственно **циклом**) **длины n** . Контур (цикл) не имеет особо выделенной вершины и может быть записан в виде последовательности (*), начиная (и заканчивая) любой его вершиной.

Путь, цепь, контур, цикл называются **простыми** (соответственно **элементарными**), если каждое их ребро (соответственно каждая вершина и каждое ребро) входит в последовательность (*) ровно один раз (не считая последней вершины в записи цикла). Можно рассматривать цепь как траекторию на графе. Тогда путь - это траектория, проходящая каждое свое ребро в направлении его ориентации. Простой путь не проходит дважды ни по одному своему ребру, элементарный путь не проходит дважды ни через одну вершину. В последовательности (*) ребро может повторяться только тогда, когда в ней повторяется некоторая вершина или когда последовательность (*) имеет такой вид:

$$v_{i_0} e_1 v_{i_1} e_1 v_{i_0} \quad (**)$$

Элементарные путь, цепь, контур, цикл можно считать просто некоторыми подграфами графа G .

Для графа, изображенного на рис. 1, цепь $v_1 e_1 v_2 e_3 v_4 e_5 v_3 e_4 v_4 e_6 v_5$ является простой, но не элементарной (вершина v_4 встречается дважды), и не является путем (ребра e_1 и e_6 имеют противоположные направления); $v_4 e_3 v_2 e_2 v_3 e_5 v_4$ - элементарный контур.

Если цепь $[v_{i_0}, v_{i_n}]$, отличная от (**), не является элементарной, то некоторая вершина v входит в последовательность (*) хотя бы дважды. Часть последовательности (*) между этими двумя вхождениями вершины v_{i_k} есть цикл, и после вычеркивания этой части из последовательности (*) остается цепь с теми же концами v_{i_0} и v_{i_n} . Повторяя эту операцию пока возможно, мы получим в результате элементарную цепь или цепь вида (**), которую можно заменить одной вершиной v_{i_0} . То же можно сделать с простым циклом. Тем самым устанавливаются следующие свойства цепей:

- всякая цепь $[v_1, v_2]$ содержит элементарную подцепь с теми же концами;
- простая, но не элементарная цепь содержит элементарный цикл;
- простой цикл, проходящий через ребро e (вершину v), содержит

элементарный подцикл, проходящий через e (v).

Таким же способом устанавливаются аналогичные утверждения для пути и для контура.

2. Легко видеть, что для любого графа отношение между вершинами “быть связанными цепью” есть транзитивное замыкание отношения соседства; оно является рефлексивным (вершина сама представляет собой цепь нулевой длины), симметричным (направление цепи не имеет значения) и транзитивным (склейка, т.е. последовательное прохождение цепей $[a, b]$ и $[b, c]$ дает цепь $[a, c]$). Таким образом, это отношение есть отношение эквивалентности. Вершины графа разбиваются на классы эквивалентности. Подграфы, натянутые на эти классы вершин, называются **компонентами связности** графа (или **связными компонентами**). Любые две вершины из одной связной компоненты соединены хотя бы одной цепью, вершины же из разных компонент не связаны цепью. Граф на рис. 2.1 имеет 3 связные компоненты, одна из них состоит из одной изолированной вершины v_6 .

Граф называется **связным**, если он имеет ровно одну компоненту связности, т.е. если любые две его вершины связаны цепью. Компоненты связности любого графа G являются максимальными (по включению) связными подграфами графа G , т.е. если к связной компоненте добавить некоторое множество вершин и/или ребер, ей не принадлежащих, то полученный подграф будет несвязным. Для решения многих задач достаточно рассматривать только связные графы в том смысле, что задача для несвязного графа сводится к рассмотрению его отдельных компонент связности.

Чтобы проверить, что граф (заданный перечнем ребер или одной из матриц) связан, достаточно построить связную компоненту произвольно выбранной его вершины. Для этого, перебирая множество ребер, присоединим к этой вершине сначала смежные с ней, затем - смежные с присоединенными и т. д., пока этот процесс можно продолжать. Если все вершины графа окажутся присоединенными, граф связан; иначе - подграф, натянутый на построенное множество вершин, образует связную компоненту, и можно продолжить процесс с оставшимися вершинами, чтобы выделить все компоненты связности.

На множестве вершин V связного графа можно ввести **расстояние** как минимальное число ребер $d(v_1, v_2)$ в цепи, связывающей v_1 и v_2 . Очевидно, минимум достигается на элементарных цепях. Для $d(v_1, v_2)$ выполнены все свойства расстояния:

- 1) $d(v_1, v_1) = 0$, $d(v_1, v_2) > 0$ при $v_1 \neq v_2$;
- 2) $d(v_1, v_2) = d(v_2, v_1)$;

3) $d(v_1, v_2) + d(v_2, v_3) \geq d(v_1, v_3)$ (неравенство треугольника).

Таким образом, V образует метрическое пространство.

Замечание. Для некоторых задач ребрам графа приписывают числовые значения, называемые весом или длиной. В этих случаях длиной цепи называют сумму длин входящих в нее ребер. Для так определенной длины цепи расстояние между вершинами есть длина кратчайшей цепи.

§3. Деревья

1. Ребро e произвольного графа G называется **циклическим**, если оно принадлежит хотя бы одному элементарному циклу в графе, и **ациклическим** в противном случае. Примером ациклического ребра является висечье. Справедливы два простых утверждения:

1) при удалении из связного графа циклического ребра граф остается связным;

2) при удалении ациклического ребра граф становится несвязным.

Первое подтверждается возможностью для любой цепи заменить циклическое ребро цепью из остальных ребер цикла. Второе доказывается от противного: если бы граф оставался связным, то концы удаленного ребра были бы связаны элементарной цепью; возвратив удаленное ребро, получили бы элементарный цикл вопреки тому, что ребро ациклическое.

Связный граф без циклов называется **деревом**.

Иначе говоря, дерево - это граф, все ребра которого ациклические. Имеется несколько эквивалентных определений дерева, устанавливающих ряд его характеристических свойств, подробнее рассмотренных ниже:

1) связный граф, который становится несвязным при удалении любого ребра;

2) связный граф, у которого число ребер на единицу меньше числа вершин;

3) граф, любые две вершины которого связаны единственной элементарной цепью;

4) граф без циклов, в котором после добавления ребра, связывающего две любые вершины, появляется цикл.

2. Выберем в дереве произвольную вершину α_0 и назовем ее корнем, или вершиной нулевого яруса. Соседние вершины назовем вершинами первого яруса и т.д. - вершины, соседние вершинам $(i - 1)$ -го яруса, не отнесенные еще ни к одному ярусу, назовем вершинами i -го яруса. Каждая вершина дерева принадлежит ровно одному ярусу.

Очевидно, что номер яруса совпадает с расстоянием между его вершинами и корнем дерева. В силу свойства (3) каждая вершина i -го яруса связана ребром ровно с одной вершиной $(i - 1)$ -го яруса и не связана ребром ни с какой вершиной i -го яруса (рис. 2.5). Такое представление дерева часто является удобным. Оно показывает, в частности, что в конечном дереве всегда существуют концевые вершины

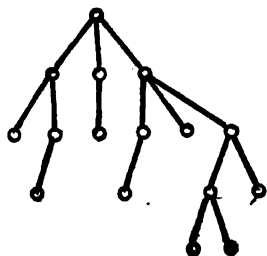


Рис. 2.5

(например, вершины последнего яруса, но, возможно, и другие).

Выделение корня в дереве D определяет на множестве его вершин частичный порядок: $\alpha < \beta$, если $\alpha \neq \beta$ и элементарная цепь из корня в вершину β содержит α (рис. 2.6). Корень α_0 является единственным минимальным элементом в этом частично упорядоченном множестве вершин, а все концевые вершины (исключая корень: он может быть, но может и не быть

концевой вершиной) - максимальными. Каждая вершина α однозначно определяет корневое поддерево D_α , натянутое на множество таких вершин β , что $\alpha \leq \beta$. В каждом таком поддереве D_α (в том числе в $D_{\alpha_0} = D$)

можно выделить вершины, непосредственно подчиненные корню поддерева, т.е. такие вершины γ_i , что $\alpha < \gamma_i$ и не существует промежуточных вершин δ_i , для которых $\alpha < \delta_i < \gamma_i$. Для каждой такой вершины γ_i определим ветвь $D_\alpha \gamma_i$ дерева D_α как корневое поддерево с корнем α , натянутое на множество вершин $\{\alpha\} \cup \{\delta: \delta \geq \gamma_i\}$. Все поддерево D_α получается из

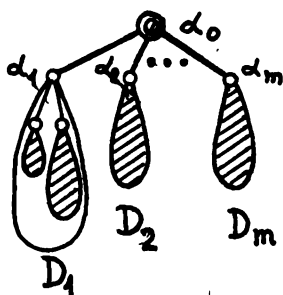


Рис. 2.6

своих ветвей склеиванием в корне α .

3. В связном графе G будем последовательно удалять циклические ребра до тех пор, пока это возможно, т.е. пока не останется ни одного циклического ребра. Мы придем к связному подграфу графа G с тем же множеством вершин, но без элементарных циклов, т.е. к дереву, называемому **остовом графа G** . Остов выбирается, вообще говоря, неоднозначно, однако все ациклические ребра обязательно входят в любой остов. Относительно остова D все ребра подграфа $G \setminus D$ называются **хордами**. Каждая хорда связывает две вершины остова.

На рис. 2.7а - пример остова (его ребра выделены) графа с 11 вершинами и 18 ребрами. Последовательность циклов и удаляемых из них ребер представлены в таблице 2.1. Оставшиеся ребра, образующие остова: 2, 3, 4, 7, 10, 11, 12, 15, 17, 18.

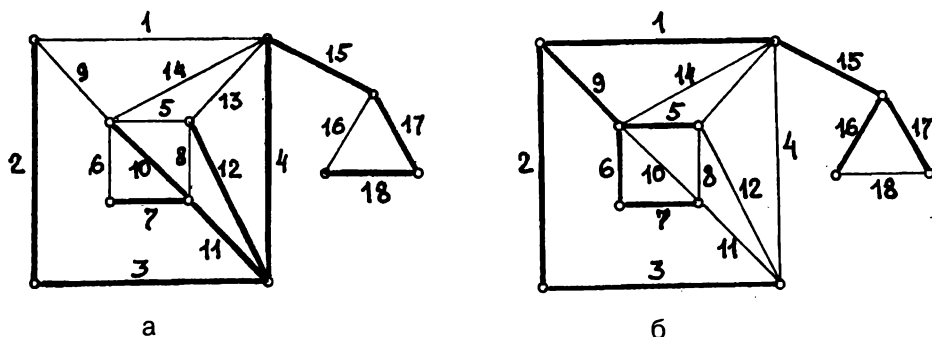


Рис. 2.7

Таблица 2.1

Цикл	Удаляемое ребро
[1, 9, 14]	1
[6, 7, 10]	6
[5, 13, 14]	5
[4, 12, 13]	13
[8, 11, 12]	8
[2, 9, 14, 4, 3]	9
[14, 10, 11, 4]	14
[16, 17, 18]	16

Удобнее строить остова графа следующим образом. Упорядочив произвольно множество ребер графа, будем, начиная с первого из них, последовательно добавлять ребра, не образующие цикла вместе с какими-нибудь из ранее выбранных. Для этого достаточно выбирать очередное ребро e_i так, чтобы одна из его вершин была инцидентна

какому-либо из ребер $e_1, \dots, e_{i-1}$, а другая - не инцидентна никакому из них. При этом всякий раз мы будем добавлять к строящемуся дереву новое концевое ребро.

Для того же графа на рис.2.76 - остов, построенный аналогичным образом. Последовательность присоединяемых ребер, составляющих в конечном счете остов, - в первом столбце таблицы 2.2. Во втором столбце - ребра, не включаемые в остов на данном этапе построения: комментарий - в третьем столбце. Обратите внимание на то, что ребра, первоначально не присоединенные, могут впоследствии войти в остов (таковы ребра 5, 6, 7).

Ребро 8 отвергается дважды по разным причинам. Ребра 4, 8, 10, 11, 12, 13, 14, 18 - хорды.

Очевидно, что удаление из дерева концевого ребра вместе с концевой вершиной приводит к связному графу без элементарных циклов, т.е. к дереву, содержащему на одну вершину и на одно ребро меньше, чем исходное дерево. Продолжая этот процесс, мы придем (если исходное дерево конечно) к дереву, состоящему из одного ребра (и двух его вершин). Поскольку из исходного дерева удалено одинаковое число ребер и вершин, то можно сделать вывод:

в любом конечном дереве число ребер на единицу меньше числа вершин.

Обратно, пусть связный граф G с b вершинами содержит $(b - 1)$ ребер. Докажем, что G - дерево в соответствии с характеристическим свойством (2). Действительно, в противном случае, удалив некоторое число q циклических ребер, мы получили бы остов с b вершинами и $(b - 1 - q)$ ребрами, что невозможно (так как остов - дерево).

Если к дереву D добавить ребро (α, β) , где $\alpha, \beta \in D$, то в графе появится (единственный!) элементарный цикл, составленный из этого ребра и (единственной) элементарной цепи $[\alpha, \beta] \subseteq D$. Если удалить из дерева D произвольное ребро (γ, δ) , не удаляя вершин, то получится несвязный граф (так как любое ребро дерева - ациклическое), состоящий ровно из двух компонент связности (так как восстановление этого ребра превращает граф в связный). Любой подграф H дерева не содержит элементарных циклов, поэтому все компоненты связности подграфа H - деревья..

!

Таблица 2.2

Ребра остова		Комментарий
1		
2		
3		
	4	образует цикл с ребрами 1, 2, 3
	5	не смежно ни одному из ранее выбранных ребер
	6	-- " -- " -- " --
	7	-- " -- " -- " --
	8	-- " -- " -- " --
9		
5		
6		
7		
	8	образует цикл с ребрами 5, 6, 7
	10	образует цикл с ребрами 6, 7
	11	образует цикл с ребрами 2, 3, 6, 7, 9
	12	образует цикл с ребрами 2, 3, 5, 9
	13	образует цикл с ребрами 1, 9, 5
	14	образует цикл с ребрами 1, 9
15		
16		
17		
	18	образует цикл с ребрами 16, 17

§4. Игра двух лиц с открытой информацией

Как пример применения понятий графа и дерева рассмотрим дискретную игру двух лиц А и В с открытой информацией.

Игра представляет собой:

- множество ситуаций $H = \{H_i\}$ (среди них - начальная ситуация H_0);
- правила игры, определяющие возможные переходы из одной ситуации в другую: для каждой ситуации H_i определено множество $T_i = \{H_{i_1}, H_{i_2}, \dots, H_{i_k}\}$ ситуаций, в которые можно перейти ходом одного из игроков;

- для некоторых ситуаций H_i множество T_i пустое - такие ситуации называются заключительными; каждой из них приписан один из символов А или В, называемых результатом игры (возможен вариант: один из трех символов А, В, 0).

Партия (тур) игры состоит в том, что игроки по очереди (будем

считать, что первый ход принадлежит игроку А) делают допустимые правилами ходы и, начиная с ситуации H_0 , переводят игру в очередную ситуацию. При попадании в любую заключительную ситуацию игра заканчивается, и результат определяет, какой из игроков - А или В - выиграл в этой партии (результат 0, если он является допустимым, означает ничью).

Дискретная игра может быть представлена оргграфом $\{H, T\}$: H - множество вершин, $T(H_i)$ задает множество дуг, исходящих из H_i . Партия (тур) представляет собой траекторию с началом в H_0 и концом в одном из заключительных состояний; однако возможен вариант, когда партия бесконечна, если, например, в графе имеется контур. В дальнейшем будем рассматривать только конечные игры.

Примером конечной игры (с возможностью ничейного исхода) являются шахматы. Правила шахматной игры определяют допустимые ходы и обеспечивают конечность игры: ничья после трехкратного повторения позиции, а также после 50 ходов без взятий, превращений и ходов пешек.

Отметим, кстати, что позиция фигур на шахматной доске (даже с указанием очередности хода) не определяет, вообще говоря, ситуации полностью; иногда необходимо знание предыстории партии: двигался ли король - для возможности рокировки и др.

Для исследования игры удобно использовать развернутую форму ее графа - в виде корневого дерева. H_0 - корень дерева; ситуации $T(H_0)$ соответствует множество вершин 1-го яруса; для каждой вершины множество $T(\alpha)$ составляет множество соседних с ней вершин 2-го яруса и т.д. В вершинах четных ярусов (0, 2, 4,...) ход принадлежит 1-му игроку (А), в нечетных (1, 3, 5,...) - 2-му (В). При таком представлении игры одна и та же ситуация соответствует многим вершинам, если в нее можно попасть из H_0 различными путями: в дереве в каждую вершину ведет единственный путь из H_0 , определяемый начальным отрезком партии до этой ситуации. Для конечной игры дерево имеет конечное число ярусов, и все концевые вершины (они могут находиться на разных ярусах) соответствуют заключительным ситуациям: им приписаны значения А или В (или 0 при возможности ничейного исхода). Длину максимального пути назовем длиной игры.

Существенное свойство рассматриваемого нами типа игр: игрок при своем ходе знает, в какой ситуации он находится и в какие ситуации ведет каждый из допустимых ходов. К такому типу игр не относится большинство карточных игр и домино, поскольку игроку обычно не известен расклад; тем самым при выборе хода он не знает текущей ситуации. По другой причине к этому типу игр не относятся, например,

нарды или лото, т. к. возможность выбора хода зависит от результата стохастического эксперимента - бросания игральные костей.

Стратегия f игрока A - это некоторое соответствие $f(H_i) = H'_i \in T_i$, назначающее для каждой ситуации H_i , в которой может оказаться игрок A , один определенный ход. Аналогично определяется стратегия игрока B .

Если выбрана стратегия f игрока A и стратегия g игрока B , то тем самым определена партия (f, g) , поскольку в каждой не заключительной ситуации однозначно определен переход в следующую ситуацию. Исход игры определяется заключительной ситуацией, в которую приходит партия.

Выбор стратегии игроком A (соответственно, игроком B) означает указание для каждой вершины четного (соответственно, нечетного) яруса ровно одной исходящей дуги. Выбор пары стратегий (f, g) выделяет ровно один путь из H_0 в одну из заключительных вершин.

Стратегия f игрока A называется **выигрышной** (соответственно, **беспроеигрышной**), если для любой стратегии g игрока B партия (f, g) заканчивается выигрышем игрока A (соответственно, выигрышем или ничьей). Выигрышная и беспроеигрышная стратегии игрока B определяются симметрично.

Замечание. Выигрышная стратегия может быть только у одного из игроков. Беспроеигрышная стратегия может быть как у одного, так и у обоих игроков.

Чтобы проиллюстрировать понятие стратегии рассмотрим игру НИМ. На столе лежат N спичек. Игрокам разрешается по очереди удалять 1, 2 или 3 спички. Проигрывает тот, кто удаляет последнюю спичку.

Примерами стратегий начинающего игрока могут быть такие:

- при каждом ходе брать 1 спичку;
- при первом ходе взять 2 спички, а затем брать столько же, сколько взял второй игрок при предыдущем ходе, пока на столе больше 5 спичек; далее - брать 1 спичку.

При $N = 20$ начинающий игрок обладает выигрышной стратегией: взять при первом ходе 3 спички, и в дальнейшем, при своем ходе брать $(4 - b)$ спичек, где b - число спичек, взятых игроком B на предыдущем ходе. При такой стратегии после хода игрока A на столе будет оставаться последовательно 17, 13, 9, 5, 1 спичек, и игрок B вынужден взять последнюю.

Если же $N = 25$, то выигрышная стратегия имеется у игрока B : брать всегда $(4 - a)$ спичек, где a - число спичек, взятых игроком A на предыдущем ходе. Тогда после его хода на столе будет оставаться последовательно 21, 17, 13, 9, 5, 1 спичек, и последнюю спичку берет игрок A .

Для конечной игры справедливы две следующих теоремы.

Теорема 2. В игре, допускающей ничейный исход, хотя бы один из игроков обладает беспроеигрышной стратегией.

Современный Гуманитарный Университет

Доказательство теоремы 1 будем вести индукцией по длине игры t .

Если $t = 1$, то дерево игры имеет вид, изображенный на рис. 2.9а. Возможны два варианта:

- среди ситуаций $H_1, \dots, H_k$ (все они - заключительные) имеется ситуация H_i , помеченная символом А, означающим выигрыш игрока А; в этом случае начинающий имеет выигрышную стратегию $f(H_0) = H_i$, т.е. выбор хода из H_0 в H_i ;

- все ситуации $H_1, \dots, H_k$ помечены символом В; тогда выигрышная стратегия (пустое множество ходов) - у игрока В: начинающий сам при любом своем ходе попадает в заключительную позицию с выигрышем игрока В.

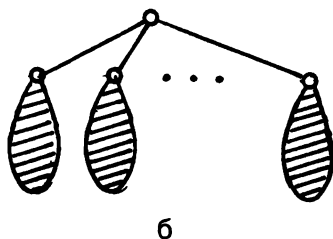
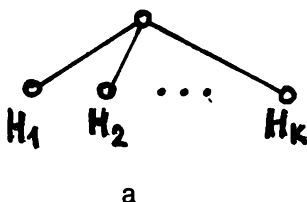


Рис. 2.9

Пусть теперь $t > 1$ (рис. 2.9б). Если удалить корень и исходящие из него дуги, то останется некоторое количество корневых поддеревьев, в каждом из которых корнем является вершина 1-го яруса первоначального дерева. Эти поддеревья представляют варианты игры, возникающие после того или иного первого хода игрока А, но начинающим в каждой из ветвей является уже игрок В. Все эти подыгры имеют длину меньше t ; поэтому, по предположению индукции, в каждой из них у одной из сторон имеется выигрышная стратегия. Возможны два варианта:

- во всех подыграх выигрышная стратегия у начинающего игрока (В); тогда в исходной игре выигрышная стратегия у игрока В: какой бы первый ход ни выбрал игрок А, его противник В может выбрать свою выигрышную стратегию в полученной подыгре;

- хотя бы в одной из подыгр есть выигрышная стратегия у не начинающего игрока (т.е. А); тогда в исходной игре выигрышная стратегия - у игрока А: она состоит в том, чтобы первым ходом попасть именно в эту подыгру, в ней выбрать свою выигрышную стратегию; в остальных подыграх можно выбрать стратегию произвольно - все равно партия в эти подыгры не переходит. Теорема доказана.

Доказательство теоремы 2 проводится таким же образом. При этом может оказаться, что беспроеигрышная стратегия имеется у обоих игроков.

Замечание. Для шахматной (а также шашечной) игры справедлива теорема 2, т.е. либо белые, либо черные, либо обе стороны имеют беспроеигрышную стратегию. Однако в настоящее время не известно, у какой именно стороны имеется беспроеигрышная стратегия и имеется ли у какой-либо из сторон выигрышная стратегия.

§5. Эйлеровы (четные) графы. Цикломатическое число

1. Здесь мы будем рассматривать подграфы, может быть, несвязные, содержащие все вершины графа. Пусть G - граф, содержащий p занумерованных ребер $e_1, e_2, \dots, e_p$. Каждому подграфу $H \subseteq G$ поставим в соответствие p -мерный вектор $(a_1, a_2, \dots, a_p)$ из нулей и единиц: $a_i = 1$, если $e_i \subseteq H$; $a_i = 0$, если $e_i \notin H$ (**характеристический вектор подграфа H**). Это соответствие, очевидно, взаимно однозначное. Более того, сумме по модулю 2 подграфов H_1 и H_2 - $H_1 \oplus H_2$ - соответствует (поразрядная) сумма по модулю 2 их характеристических векторов. Над множеством коэффициентов $\{0, 1\}$ множество всех подграфов образует линейное пространство: умножение любого подграфа H на 1 дает H , умножение на 0 - пустой подграф, т.е. подграф, не содержащий ребер и состоящий из одних изолированных вершин графа G . Нетрудно видеть, что пространство подграфов графа G и пространство характеристических векторов его подграфов изоморфны и имеют размерность p . Базисом может служить множество всех однореберных подграфов. Характеристический вектор каждого такого подграфа содержит ровно одну единицу, причем для разных подграфов - на разных местах.

2. Назовем граф **четным**, если степени всех его вершин - четные числа. Любую элементарную цепь (кроме цикла) в четном графе можно продлить ребром, не принадлежащим цепи, так как конец цепи имеет степень 1 в цепи и степень не меньше 2 в графе. Если граф конечен, то, продолжая цепь, мы через конечное число шагов вторично придем в уже пройденную вершину, т.е. часть полученной (уже не элементарной) цепи образует элементарный цикл, после удаления которого остается четный граф, так как все степени изменяются на четное число (2 - для вершин цикла и 0 - для вершин, не принадлежащих циклу). В этом графе можно снова выделить некоторый элементарный цикл и т.д., пока в графе не останется ни одного ребра. Таким образом, каждый конечный четный граф можно представить в виде суммы попарно не пересекающихся (по ребрам) элементарных циклов, откуда,

в частности, следует, что каждое его ребро циклическое.

Если конечный четный граф связан, то нетрудно показать (например, индукцией по числу его элементарных циклов), что в нем существует простой (т.е. не самопересекающийся по ребрам) цикл, содержащий все ребра графа. Такой цикл называется **эйлеровым циклом**, а обладающий им граф называется **эйлеровым графом**. Поскольку каждая вершина простого цикла имеет в нем четную степень (каждое прохождение через вершину по циклу использует два ребра, инцидентных этой вершине), то доказана следующая теорема.

Теорема Эйлера. Для того чтобы конечный связный граф был эйлеровым, необходимо и достаточно, чтобы он был четным.

В конечном несвязном четном графе все компоненты связности - эйлеровы графы.

Будем называть вершину ориентированного графа **равновесной**, если число дуг, входящих в вершину, равно числу дуг, исходящих из нее. Ориентированный граф назовем **равновесным**, если все его вершины равновесные.

Обход (неориентированного) эйлерова графа по эйлерову циклу ориентирует каждое ребро графа в направлении обхода. Ясно, что при такой ориентации эйлеров граф является равновесным. Пусть теперь дан равновесный граф. Повторяя предыдущее рассуждение (вместо циклов надо говорить о контурах), легко показать, что каждый конечный равновесный граф можно представить в виде суммы элементарных контуров, попарно не пересекающихся по ребрам (а также то, что в конечном связном равновесном графе существует простой контур, содержащий все ребра графа).

Другими словами, теорема Эйлера означает, что четность всех вершин графа есть необходимое и достаточное условие существования обхода графа, при котором каждое ребро проходится ровно один раз.

Из теоремы Эйлера можно вывести интересное следствие. В произвольном связном графе G раздвоим каждое ребро $e = (v_1, v_2)$, т.е. заменим его парой кратных ребер e' , e'' с теми же вершинами. Степень каждой вершины удвоится и станет четной. В полученном четном графе G_1 существует эйлеров цикл (а если ребра e' , e'' ориентовать в противоположных направлениях, то каждая вершина будет равновесной и существует эйлеров контур). Если рассмотреть его как обход графа G_1 , снова склеить пары e' , e'' в одно ребро, то в первоначальном графе G эта траектория будет (уже не простым) циклом, проходящим через каждое свое ребро ровно два раза (причем в противоположных направлениях). Итак, в каждом связном графе существует цикл, содержащий ровно 2 раза каждое ребро графа, что можно трактовать

как соответствующий обход лабиринта с возвращением в начало обхода.

3. Элементарный путь, проходящий через все вершины графа, называется **гамильтоновым**; если совпадают его начало и конец, то это гамильтонов цикл.

Известная задача о коммивояжере формулируется так: имеется n городов, попарные расстояния между которыми заданы. Коммивояжер желает посетить все города так, чтобы суммарная длина пути была минимальной. Это означает, что нужно найти гамильтонов путь минимальной длины в полном графе K_n , ребрам которого приписаны положительные числа - длины ребер (для варианта задачи - обхода с возвращением в начало пути - ищется минимальный гамильтонов цикл).

Последовательность перестановок, описанная в гл. I, §3, может быть интерпретирована следующим образом. Рассмотрим граф G_n , вершины которого соответствуют всем $(n!)$ перестановкам множества $\{1, 2, \dots, n\}$, а ребра связывают пары вершин, отличающиеся транспозицией соседних элементов (таким образом, каждая вершина соединена с $(n - 1)$ другими вершинами). Тогда указанная последовательность соответствует гамильтонову пути в графе G_n . На рис. 2.10а,б - соответствующие иллюстрации для $n = 3$ и $n = 4$.

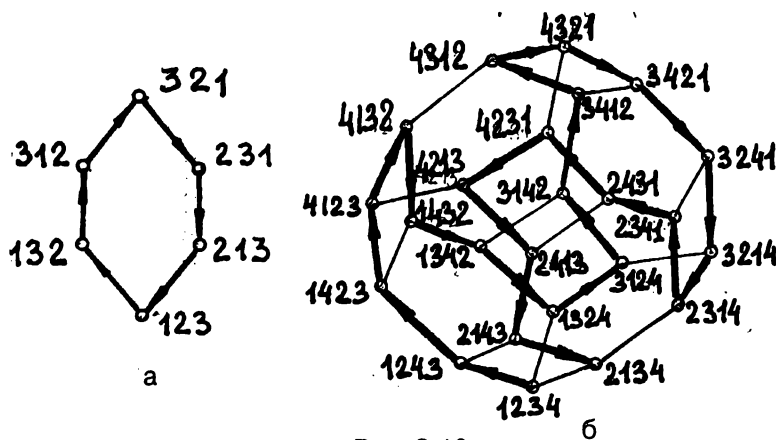


Рис. 2.10

4. Сумма двух четных подграфов любого графа есть также четный подграф. Если s_1 и s_2 - степени некоторой вершины в подграфах H_1 и H_2 соответственно, а s_{12} - ее степень в пересечении $H_1 \cap H_2$, то степень $s(\alpha)$ вершины α в подграфе $H_1 \oplus H_2$ равна $(s_1 - s_{12}) + (s_2 - s_{12}) = s_1 + s_2 - 2s_{12}$ (кстати, это пример применения комбинаторной формулы включения-...

исключения). Если s_1 и s_2 - четные, то $s(\alpha)$ также четное. Поэтому четные подграфы образуют подпространство в пространстве всех подграфов, которое совпадает с подпространством, натянутым на множество всех элементарных циклов.

Перейдем к выяснению размерности v этого подпространства. Пусть G - связный граф с p ребрами и b вершинами, D - некоторый его остов. Число хорд равно $(p - b + 1)$. Каждая хорда (α, β) вместе с единственной элементарной цепью $[\alpha, \beta] \subseteq D$ образует элементарный цикл, причем векторы всех таких циклов (для разных хорд) образуют независимую систему Σ , так как каждый из циклов содержит ребро (свою хорду), не принадлежащее ни одному из остальных циклов системы. Отсюда $v \geq p - b + 1$.

С другой стороны, каждый четный подграф, в частности любой простой цикл выражается через циклы системы Σ . Действительно, прибавим к четному подграфу H те циклы из Σ , которые содержат хорды графа, принадлежащие H . Полученная сумма не будет содержать ни одной хорды (каждая хорда входит в сумму дважды: в подграф H и в один из циклов системы Σ), т.е. будет некоторым подграфом дерева D , откуда следует, что это пустой подграф, так как в противном случае четный подграф (сумма H и циклов), имеющий элементарные циклы, содержался бы в дереве D . Таким образом, $v = p - b + 1$.

Для несвязного графа с k компонентами связности базис пространства четных подграфов получается объединением базисов его связных компонент, а число ребер и вершин суммируется. Поэтому, если i -я компонента содержит p_i ребер и b_i вершин, то $v = p - b + k$, где

$$p = \sum_{i=1}^k p_i, \quad b = \sum_{i=1}^k b_i. \quad \text{Число } v \text{ называется } \mathbf{цикломатическим \text{ числом}}$$

графа. Ввиду того, что $v > 0$, для любого графа справедливо неравенство $k \geq b - p$. Деревья - это связные графы с цикломатическим числом, равным 0.

Примеры. 1) Для полного графа K_5 (5 вершин, $C_5^2 = 10$ ребер) цикломатическое число $v = 10 - 5 + 1 = 6$.

2) Для графа на рис. 7 имеем $p = 18$, $b = 11$, $k = 1$, откуда $v = 18 - 11 + 1 = 8$.

3) Рассмотрим соотнесенный неориентированный граф для графа на рис. 10б. Число его вершин равно числу перестановок из 4 элементов: $4! = 24$. Степени всех вершин равны 3. Поэтому число ребер графа $p = 24 \cdot 3 / 2 = 36$. Граф связан, т.е. $k = 1$. Цикломатическое число равно $v = 36 - 24 + 1 = 13$.

§6. Двухполюсные сети. Потоки в сетях

1. Граф, некоторые вершины которого выделены, называется **сетью**. Выделенные вершины называются **полюсами** сети. Например, дерево с корнем можно рассматривать как однополюсную сеть. Вершины, отличные от полюсов, называются **внутренними вершинами** сети. Ребро, инцидентное хотя бы одному полюсу, называется **полюсным ребром**; остальные ребра называются **внутренними**.

(k, l) -**полюсником** называется сеть с $(k + l)$ полюсами, разбитыми на два класса: k входных и l выходных полюсов. $(1, 1)$ -полюсник называется также двухполюсной сетью. В этом параграфе мы будем рассматривать, главным образом, двухполюсные сети без петель и без изолированных внутренних вершин (кратные ребра допускаются). Они будут называться просто сетями. Будем также называть **цепью** (без указания концов) элементарную цепь между полюсами сети - в других случаях будем обязательно указывать концы цепи или называть ее цепочкой. Обозначим входной и выходной полюсы сети S символами α_s и β_s . Полюсные ребра образуют входную и выходную звезды Z_{α_s} и Z_{β_s} , пересечение которых состоит из сквозных ребер (оно может быть пустым), инцидентных обоим полюсам.

Пусть A и B - две цепи сети S , не имеющие общих внутренних вершин, α и β - внутренние вершины этих цепей, C - элементарная цепочка $[\alpha, \beta]$, не пересекающаяся с A и B нигде, кроме своих концов (рис. 2.11). Тогда существуют две цепи, проходящие через цепочку C в разных направлениях: $\alpha_s A_1 \alpha C \beta B_2 \beta_s$ и $\alpha_s B_1 \beta C \alpha A_2 \alpha_s$. Подграф, образованный парой таких цепей A и B и связывающей их цепочкой C , называется мостиком.

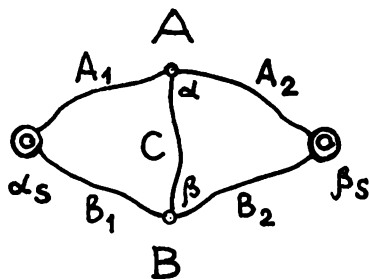


Рис. 2.11

2. Рассмотрим две операции над сетями. Пусть S_1 - сеть с полюсами α_1 и β_1 , S_2 - сеть с полюсами α_2 и β_2 , причем S_1 и S_2 не имеют общих элементов. Сеть $S_1 \cdot S_2$ с полюсами α_1 и β_2 , полученная склеиванием вершин β_1 и α_2 , называется **последовательным соединением** сетей S_1 и S_2 (рис. 2.12а); сеть $S_1 \vee S_2$ с полюсами α_1 и β_1 , полученная склеиванием полюсов α_1 с α_2 и β_1 с β_2 , называется **параллельным соединением** сетей S_1 и S_2 .

5/8/14 31 paper
to Berlin

нием сетей S_1 и S_2 (рис. 2.126). Аналогично определяются последовательное и параллельное соединения большого числа сетей: $S_1 \cdot S_2 \dots S_n$ и $S_1 \vee S_2 \vee \dots \vee S_n$.

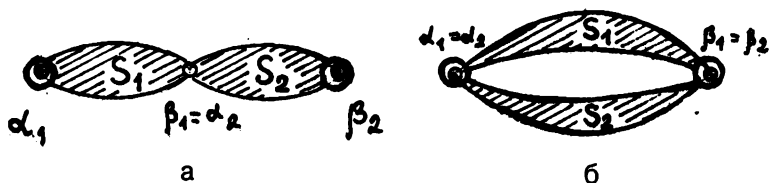


Рис. 2.12

Сеть, полученная из односторонних сетей в результате конечного числа параллельных и последовательных соединений, называется **параллельно-последовательной сетью**. Другими словами, класс параллельно-последовательных сетей (сокращенно П-сетей) определяется индуктивно:

- (1) односторонняя сеть есть П-сеть,
- (2) если S_1 и S_2 - П-сети, то $S_1 \cdot S_2$ и $S_1 \vee S_2$ - тоже П-сети.

Характеристическое свойство П-сетей: они не содержат мостиков.

3. Добавим к сети S дополнительное, *источниковое* ребро, связывающее полюсы (иногда, например, в случае ориентированных сетей бывает удобным ориентировать это ребро от выхода к входу сети).

Полученную расширенную сеть обозначим через $\hat{S}$. Заметим, что добавление источникового ребра к любой цепи A превращает ее в элементарный цикл $\hat{A}$.

Нетрудно показать, что в произвольной сети сумма по модулю 2 **нечетного** числа цепей A_i содержит цепь. В самом деле, сложим по модулю 2 циклы $\hat{A}_i$, соответствующие данным цепям A_i . В результате получим четный граф $\sum \hat{A}_i$. Источниковое ребро входит в эту сумму, так как оно участвует в сложении нечетное число раз. Очевидно, что оно является циклическим ребром в $\sum \hat{A}_i$ (напомним, что в четном графе все ребра циклические). Содержащий его цикл после удаления самого источникового ребра есть искомая цепь.

4. Пусть S - произвольная частично ориентированная сеть, каждому ребру e которой приписано неотрицательное число $c(e)$ - пропускная

способность. **Потоком в сети S** называется пара (f, ω) , где ω - некоторая ориентация всех неориентированных ребер сети, а $f(e)$ - заданная на множестве всех ребер функция с неотрицательными значениями, не превосходящая пропускных способностей, и такая, что в каждой внутренней вершине выполнен закон Кирхгофа, согласно которому сумма значений потока по ребрам, входящим в вершину, равна сумме значений потока по ребрам, исходящим из вершины. Другими словами, для $f(e)$ выполнены условия:

- (1) $0 \leq f(e) \leq c(e)$ для всех ребер сети;
- (2) $R(\alpha) = 0$ для всех внутренних вершин α , где

$$R(\alpha) = \sum_{e \in \Gamma(\alpha)} f(e) - \sum_{e \in \Gamma'(\alpha)} f(e),$$

а $\Gamma(\alpha)$ (соответственно $\Gamma'(\alpha)$) - множество всех ребер, исходящих из α (соответственно входящих в α) при ориентации ω .

Поскольку сумма значений $R(\alpha)$ по всем вершинам сети, включая полюсы, равна 0 (каждое ребро является исходящим для одной вершины и входящим для другой), то $R(\alpha_s) = -R(\beta_s)$. Значение $R = R(\alpha_s)$ называется **величиной потока**. Между прочим, сходные рассуждения показывают, что если сеть допускает поток ненулевой величины, то полюсы сети находятся в одной компоненте связности. Если сеть изображает какую-либо проводящую систему (сеть дорог, трубопровод и т.п.), то R определяет величину потока, входящего в одном полюсе и выходящего из другого, т.е. проходящего через сеть. Если на ориентированном от β_s к α_s источниковом ребре положить $f = R$ (допуская, что для этого ребра f может быть меньше 0), то закон Кирхгофа будет выполняться для всех вершин сети, а R будет определять величину циркуляции через сеть.

5. Поставим задачу определить максимальную величину потока $R_{\max}$ через сеть S при заданных значениях пропускных способностей. Ответ может быть получен в терминах сечений сети.

Сечением в сети называется множество ребер, при удалении которого сеть становится несвязной, причем полюсы попадают в разные компоненты связности. Ясно, что каждая цепь (а тем более каждый путь из α_s в β_s) проходит хотя бы через одно ребро сечения. В сети на рис. 13 примерами сечений являются $\{d, e, f\}$, $\{b, c, e, g, h\}$, $\{d, g, h, i\}$. Сечение будем называть **простым**, если при удалении любого его ребра оно перестает быть сечением. Так, $\{d, e, f\}$ и $\{b, c, e, g, h\}$ являются простыми сечениями, в то время как $\{d, g, h, i\}$ не является таковым: можно удалить ребро h или ребро i . Очевидно, что для каждого ребра простого сечения можно указать цепь, которая проходит через это ребро, но не

проходит через другие ребра данного сечения: иначе это ребро было бы излишним.

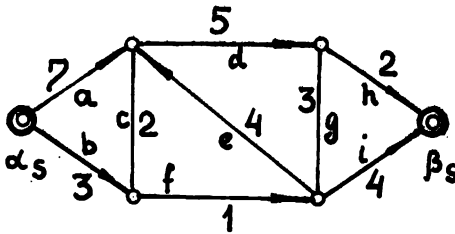


Рис. 2.13

Если в связной сети удаляется простое сечение, то сеть распадается ровно на две части: левую часть, содержащую α_s , и правую часть, содержащую β_s . Каждое ребро простого сечения связывает вершины из разных частей. Будем называть ребро сечения **прямым**, если оно в сети

не ориентировано или ориентировано слева направо, и **обратным** - в противном случае. Будет ли ориентированное ребро прямым или обратным, вообще говоря, зависит от выбора сечения. Так, в примере на рис. 2.13 ребро e в сечениях $\{d, e, f\}$ и $\{b, c, e, g, h\}$ - обратное, а в сечении $\{a, c, e, g, i\}$ - прямое.

Каждому простому сечению W припишем пропускную способность $c(W)$, равную сумме пропускных способностей всех его прямых ребер. В примере на рис. 2.13 сечение $\{d, e, f\}$ имеет пропускную способность $5 + 1 = 6$, а сечение $\{b, c, e, g, h\}$ - пропускную способность $3 + 2 + 3 + 2 = 10$. Если сеть несвязна и ее полюсы находятся в разных компонентах связности, то естественно считать (единственным) простым сечением пустое множество, а его пропускную способность нулевой.

Теорема Форда-Фалкерсона о максимальном потоке:

максимальная величина потока $R_{\max}$ через сеть S равна минимальной из пропускных способностей $c_{\min}$ ее простых сечений.

Неравенство $R_{\max} \leq c_{\min}$ можно считать достаточно очевидным из физических соображений, и в его справедливости нетрудно убедиться, просуммировав величину $R(\alpha)$ по всем вершинам α левой компоненты произвольного сечения W . Эта сумма, с одной стороны, равна $R(\alpha_s)$, а с другой стороны, она равна алгебраической сумме величин потоков, идущих через сечение слева направо (потоки по ребрам, идущим из левой компоненты в правую суммируются со знаком плюс, а по ребрам, идущим в обратном направлении, - со знаком минус). Поскольку $f(e) \leq c(e)$, отсюда следует, что $R \leq c(W)$. Ясно также, что равенство $R = c(W)$ может достигаться только, если все прямые ребра сечения W ориентированы слева направо (при ориентации ω) и для них $f(e) = c(e)$, а для всех обратных ребер $f(e) = 0$.

Несколько сложнее доказывается, что в сети S можно создать поток величины, равной $c_{\min}$.

Разработан ряд конкретных алгоритмов построения максимального потока, основанных на этой теореме.

Теорема Форда-Фалкерсона допускает обобщение на случай, когда пропускные способности приписаны не только ребрам, но и внутренним вершинам сети, и поток считается **допустимым**, если для

любой внутренней вершины α величина $\sum_{e \in \Gamma(\alpha)} f(e)$ (так же как и

величина $\sum_{e \in \Gamma'(\alpha)} f(e)$) не превосходит пропускной способности вершины

α . Теорема распространяется и на (k, l) -полюсную сеть. В обоих случаях величина максимального потока от входов к выходам сети равна минимальной пропускной способности множества элементов сети, блокирующего все цепи.

6. Наряду с определением максимальных потоков в тех или иных проводящих системах теорема Форда-Фалкерсона позволяет решать и некоторые чисто комбинаторные проблемы. Одна из них носит название задачи о назначениях. Пусть имеется N претендентов $\{\alpha_i\}$ на N вакантных должностей $\{\beta_j\}$. Возможно ли назначение всех N кандидатов, если каждый из них способен занять одну из некоторого подходящего ему подмножества V_i этих должностей? Очевидным является следующее необходимое условие такой возможности: для любой группы $k \leq N$ претендентов объединение соответствующих им множеств V_i должно содержать не меньше k должностей - иначе им не хватит мест (вспомните комбинаторный принцип Дирихле).

Ситуация может быть представлена двудольным графом, где $V_1 = \{a_i\}$, $V_2 = \{b_j\}$ и (a_i, b_j) - дуга графа, если кандидат a_i способен занять должность b_j . Если ввести две дополнительные вершины - полюсы s и t , дополнительные дуги (s, a_i) и (b_j, t) , установить подходящим образом пропускные способности вершин и ребер, то с помощью теоремы Форда-Фалкерсона можно показать, что приведенное условие является также достаточным.

§7. Кратчайшие пути и цепи

1. Есть известная задача о волке, козе и капусте. Хозяин должен переправить их с левого берега реки на правый, пользуясь лодкой, в которой кроме него может поместиться только один из трех "пассажиров".

При этом по понятным причинам нельзя в отсутствие хозяина оставлять вместе волка с козой, а также козу с капустой. В каком порядке проводить переправу? Задача довольно легко решается в уме из-за малого числа вариантов, однако рассмотрим ее с прицелом на более общий случай.

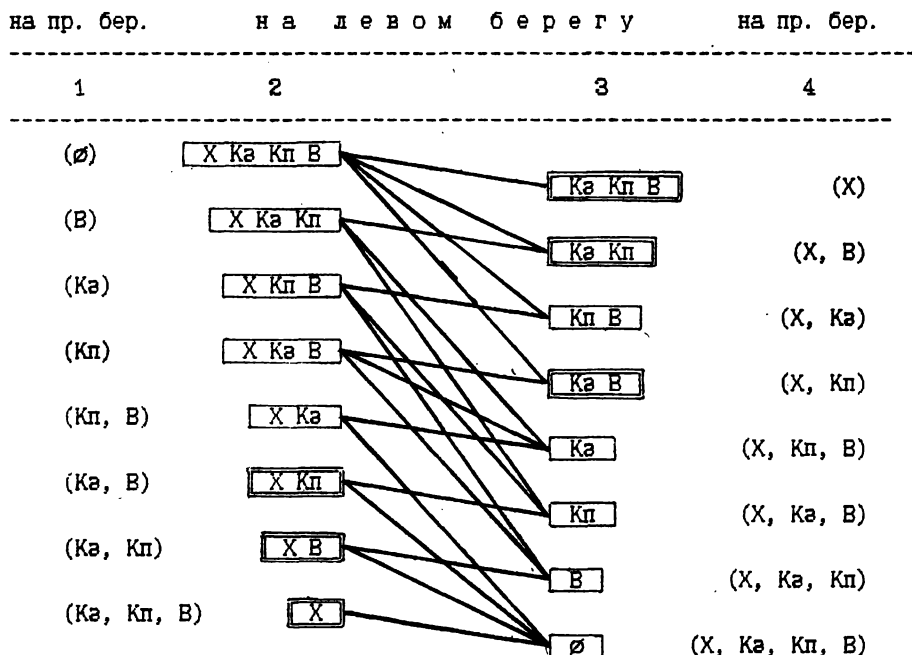


Рис. 2.14

На рис. 2.14 - сеть, вершинами которой являются всевозможные расположения на двух берегах четырех участников: X (хозяин), Кз (коза), Кп (капуста), В (волк). Каждую ситуацию, возникающую в процессе переправы, можно охарактеризовать подмножеством A персонажей, находящихся на этом этапе на левом берегу; дополнение A этого подмножества - участники, перевезенные на правый берег. Общее количество подмножеств 4-элементного множества равно $2^4 = 16$. Ребра графа связывают вершины α и β , если из ситуации α можно перейти в ситуацию β за одну ходку (переправу). Поскольку возможен и обратный рейс из β в α , то можно считать граф неориентированным.

Множество вершин естественно разбивается на 2 части: с присутствием (столбец 2 на рис. 14) или отсутствием (столбец 3 на

рис. 2.14) на левом берегу хозяина; очевидно, что граф - двудольный, т.к. при каждой переправе этот признак меняется. Например, вершина $\langle X \text{ Кз Кп} \rangle$ связана с вершинами $\langle \text{Кп} \rangle$ (если хозяин перевозит на правый берег козу), $\langle \text{Кз} \rangle$ (если перевозит капусту) и $\langle \text{Кз Кп} \rangle$ (если переправляется один). Вершина $\langle \text{Кп В} \rangle$ (хозяин с козой находятся на правом берегу) связана с вершинами $\langle X \text{ Кз Кп В} \rangle$ (если хозяин возвращается на левый берег с козой) и $\langle X \text{ Кп В} \rangle$ (если он возвращается один). В крайних столбцах рис. 14 для наглядности представлены дополнения к подмножествам-вершинам.

Задачу теперь можно сформулировать так: найти цепь из полюса $\langle X \text{ Кз Кп В} \rangle$ (все на левом берегу) в полюс $\langle \emptyset \rangle$ (все на правом берегу, на левом - никого), причем нельзя проходить через запрещенные вершины, обозначенные на рис. 14 двойной рамкой: например, ситуация $\langle \text{Кз В} \rangle$ означает опасную пару без хозяина на левом берегу, а ситуация $\langle X \text{ В} \rangle$ - на противоположном берегу коза соседствует с капустой.

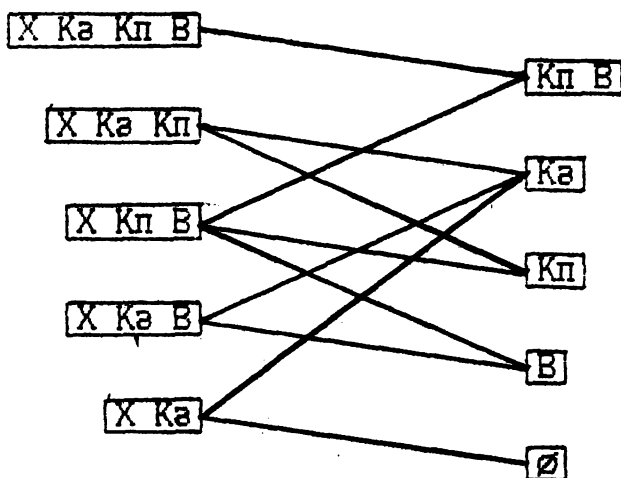


Рис. 2.15

Удалим теперь из сети запрещенные вершины вместе с инцидентными им ребрами: результат представлен на рис. 2.15. Теперь достаточно изобразить полученную сеть в более удобном виде, как на рис. 2.16. Здесь решение очевидно: обнаруживаются даже две различные элементарные цепи.

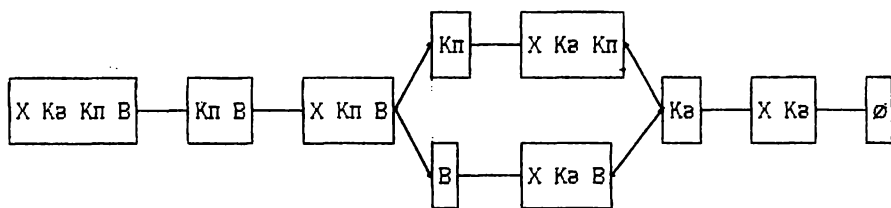


Рис. 2.16

2. Обобщение этой задачи состоит в отыскании цепи (или цепи наименьшей длины) в произвольной сети. Для достаточно большой сети наглядность уже отсутствует. Разработаны различные алгоритмы построения такой цепи, применимые к графам и сетям, заданным в различной форме.

Один из них основан на следующей конструкции. Будем присваивать вершинам сети S ранг (целочисленную метку): входному полюсу α_s - ранг 0; вершинам, смежным с полюсом α_s , - ранг 1; непомяченным вершинам, смежным с вершинами первого ранга, - ранг 2 и т.д. На k -ом этапе процесса непомяченным вершинам, смежным с вершинами $(k - 1)$ -го ранга, присваивается ранг k . Ранг вершины равен расстоянию до нее от полюса α_s . На каждом этапе вершины очередного ранга образуют как бы фронт распространения волны от полюса α_s . Процесс закончится, когда ранг будет присвоен выходному полюсу β_s (если это не произойдет, то сеть S несвязна). Кратчайшая цепь $[\alpha_s, \beta_s]$ (не единственная) определяется некоторой последовательностью вершин различных рангов: каждая вершина γ k -го ранга, начиная с β_s , связана ребром со смежной вершиной $(k - 1)$ -го ранга (например, с той, по которой γ получила свой ранг).

Для графа на рис. 2.17 представлены ранги вершин при определении расстояния и кратчайшей цепи из A в B и указана одна из них.

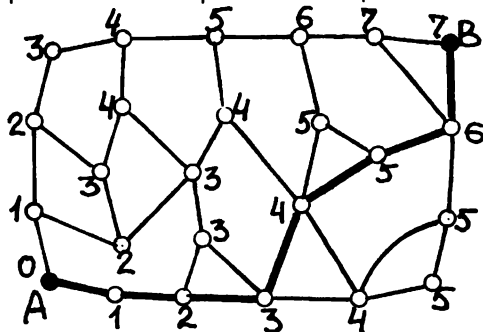


Рис. 2.17

Модификация этого алгоритма состоит в одновременном построении двух фронтов волн, исходящих из α_s и β_s : процесс заканчивается, когда эти волны встретятся.

3. Для ориентированной сети с присвоенными ее дугам длинами $\{l(e_k)\}$ алгоритм построения кратчайшего пути несколько иной. Метки $\lambda(v_i)$, присваиваемые вершинам, могут изменяться в процессе построения. Первоначально пометим полюс α_s числом 0, а каждую из остальных вершин - достаточно большим числом L (например, превышающим сумму длин всех дуг сети). Далее будем, пока возможно, производить следующее преобразование (смену) меток $\lambda(v_i)$. Если для некоторой дуги $e = (v_1, v_2)$ выполнено строгое неравенство $\lambda(v_2) - \lambda(v_1) > l(e)$, то присвоим вершине v_2 (концу дуги e) новую, меньшую метку: $\lambda(v_2) = \lambda(v_1) + l(e)$. Через конечное число таких операций для любой дуги e будет выполнено нестрогое неравенство $\lambda(v_2) - \lambda(v_1) \leq l(e)$. Тогда $\lambda(\beta_s)$ будет равно длине кратчайшего пути $[\alpha_s, \beta_s]$. Отыскать этот путь можно, двигаясь подобно предыдущему, обратным ходом из β_s в α_s по дугам $e = (v', v'')$, для которых $\lambda(v'') = \lambda(v') + l(e)$.

Пример описанного построения для сети на рис. 2.18 приведен в таблице 2.3. В первом столбце - первоначальные метки; в последующих столбцах - процесс пересчета меток: указаны только меняющиеся значения, остальные переносятся слева из предыдущих столбцов; для каждого изменения указано, по какой из вершин оно произведено. В последний столбец перенесены окончательные значения меток.

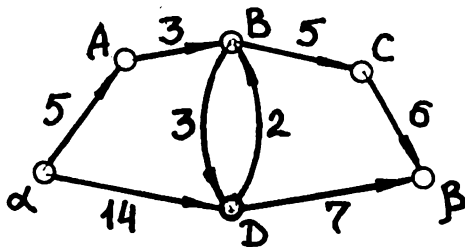


Рис. 2.18

Таблица 2.3

α	0										0
A	100	5 α									5
B	100		16 D			8 A					8
C	100			21 B				13 B			13
D	100		14 α							11 B	11
β	100					27 C	21 D		19 C		18 D

§8. Раскраска графов

1. Говорят, что вершины (соответственно ребра) неориентированного графа G без петель **правильно раскрашены**, если каждой вершине (соответственно каждому ребру) графа сопоставлен ровно один из заданного набора цветов, причем любым двум соседним вершинам (соответственно смежным ребрам) сопоставлены разные цвета. Граф, вершины или ребра которого правильно раскрашены, будем называть **правильно раскрашенным**, если из контекста ясно, какие именно элементы графа раскрашиваются или это безразлично. Очевидно, что всякий подграф правильно раскрашенного графа также правильно раскрашен. Удобно нумеровать цвета натуральными числами $1, 2, \dots$. Отрезок натурального ряда от 1 до k включительно будем обозначать N_k . Можно сказать, что правильная раскраска вершин (соответственно ребер) графа - это функция с натуральными значениями, определенная на множестве вершин (соответственно ребер) и *разнозначная* на концах каждого ребра (соответственно на ребрах каждой звезды) графа.

Основная задача состоит в определении наименьшего числа $\chi(G)$ (соответственно $\chi'(G)$) различных цветов, с помощью которых можно правильно раскрасить вершины (соответственно ребра) графа G . $\chi(G)$ называется **хроматическим числом**, а $\chi'(G)$ - **хроматическим классом** графа G .

2. При раскраске вершин будем рассматривать графы без петель и кратных ребер, поскольку наличие или отсутствие кратных ребер никак не влияет на правильность раскраски графа. Отметим, что подграф, натянутый на множество вершин, окрашенных одинаково, при правильной раскраске не содержит ребер.

Очевидно, что для всякого графа с b вершинами $\chi(G) \leq b$, причем равенство достигается только для полного графа: всякий другой граф можно раскрасить меньшим числом цветов, окрасив две несмежные вершины одинаково. С другой стороны, если в графе имеется хотя бы одно ребро, то $\chi(G) \geq 2$.

Графы, для которых $\chi(G) = 2$, называются **бихроматическими**. Ясно, что бихроматический граф является двудольным, и наоборот, всякий двудольный граф, содержащий хотя бы одно ребро, бихроматичен. Справедливо и другое условие бихроматичности и тем самым двудольности графа.

Теорема. Для того чтобы граф, содержащий хотя бы одно ребро, был бихроматическим, необходимо и достаточно, чтобы он не содержал элементарных циклов нечетной длины.

Необходимость. Поскольку при правильной раскраске цвета на цикле должны чередоваться, вершины цикла нечетной длины не могут быть раскрашены в два цвета, и, следовательно, граф, содержащий этот цикл, не может быть бихроматическим.

Достаточность. Заметим прежде всего, что дерево бихроматично. В самом деле, выберем произвольно корень и используем представление дерева, описанное в §3. Окрасим все вершины четных ярусов в один цвет, а все вершины нечетных ярусов в другой цвет. Эта раскраска правильная, так как ребра в дереве соединяют только вершины соседних ярусов.

Рассмотрим теперь элементарную цепь, соединяющую в дереве вершину i -го яруса и вершину j -го яруса. На каждом ее ребре происходит изменение номера яруса на единицу. Поэтому ее длина имеет ту же четность, что и число $i - j$. В частности, вершины, принадлежащие ярусам одинаковой четности, связаны элементарной цепью четной длины.

Если мы выберем теперь в графе G , не содержащем элементарных циклов нечетной длины, любой остов, то хорды относительно этого остова будут связывать вершины из ярусов разной четности (иначе в графе G появился бы элементарный цикл нечетной длины), т.е. двухцветная правильная раскраска остова будет правильной раскраской и в графе G .

3. Если граф содержит в качестве подграфа t -вершинный полный граф K_t , то его хроматическое число, очевидно, не меньше t . Противоположное утверждение, однако, неверно. Существуют графы даже без треугольников K_3 , имеющие сколь угодно большое хроматическое число. Пример такого графа с хроматическим числом 4 приведен на рис. 2.19.

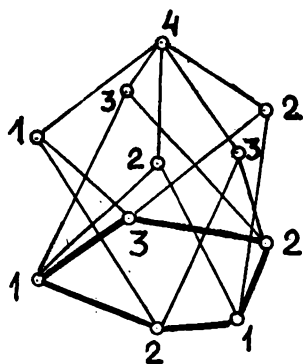


Рис. 2.19

Рассмотрим, как связано хроматическое число со степенями вершин графа. Обозначим через $s(G)$ максимальную из степеней вершин графа G , а через $\Gamma(s)$ - класс графов без кратных ребер, для которых $s(G) \leq s$. Нетрудно показать индукцией по числу вершин, что если $G \in \Gamma(s)$, то $\chi(G) \leq s + 1$. В самом деле, если число вершин в графе меньше или равно $(s + 1)$, то утверждение тривиально. Пусть теперь оно установлено для всех графов из $\Gamma(s)$, имеющих меньше вершин, чем граф G . Удалим из G звезду произвольной вершины α и в соответствии с предположением индукции правильно раскрасим граф $G \setminus Z\alpha$ $(s + 1)$ цветами. В графе G у вершины α не более s соседних вершин, поэтому в множестве цветов N_{s+1} есть

хотя бы один цвет, которым не окрашена ни одна из вершин, соседних с α . Его можно использовать для окраски вершины α , после чего граф G будет правильно раскрашен $(s + 1)$ цветами.

Полный $(s + 1)$ -вершинный граф K_{s+1} представляет собой пример графа из $\Gamma(s)$, для которого хроматическое число равно $s + 1$. Оказывается, что этот пример - единственный, а именно справедлива следующая теорема

Теорема. Пусть $s \geq 3$. Если $G \in \Gamma(s)$ и $G \neq K_{s+1}$, то $\chi(G) \leq s$.

ГЛАВА III. ЭЛЕМЕНТЫ ТЕОРИИ КОДИРОВАНИЯ

§ 1. Алфавитное кодирование

1. Кодом называется система условных знаков (символов) для передачи, обработки и хранения (запоминания) различной информации. Предметом исследования теории кодирования являются отображения конечных или счетных множеств объектов произвольной природы в множества последовательностей из цифр $0, 1, \dots, r-1$, где r – некоторое целое положительное число (в частности, $r = 2$). Такие отображения называются **кодированиями**.

Большинство задач теории кодирования укладывается в следующую схему. Для заданного множества объектов рассматривается класс кодирований, обладающих определенными свойствами. Требуется построить кодирование из рассматриваемого класса, оптимальное в некотором заранее заданном смысле. Обычно критерий оптимальности кодирования так или иначе связан с минимизацией длин кодов, в то время как требуемые свойства кодирований могут быть весьма разнообразными. Среди таких свойств: существование однозначного обратного отображения (**декодирования**), возможность исправления при декодировании ошибок различного типа, возможность простой реализации (простота алгоритма) кодирования и декодирования и т. п.

Пусть A – произвольный алфавит. Элементы алфавита A будем называть **буквами**, а конечные последовательности (кортежи), составленные из букв, – **словами в A** . **Длина** (число букв) слова α обозначается через $\lambda(\alpha)$, а слово длины 0 (**пустое слово**) обозначается символом Λ . **Соединение** слов α_1 и α_2 обозначим через $\alpha_1\alpha_2$, а соединение n одинаковых слов α – через α^n ($\alpha^0 = \Lambda$).

Пусть U – произвольное множество слов в A . Через U^n ($n = 0, 1, \dots$) обозначим множество всех слов в A^n , представимых в виде соединения n слов из U . В частности, через A^n ($n = 0, 1, \dots$) обозначим множество всех

слов длины n в алфавите A , а через A^* – множество всех слов произвольной длины в A .

Множество $U^* = \bigcup_{n=0}^{\infty} U^n$ – не всевозможные слова в алфавите A ;

это зависит от выбранного множества U . Так, если U состоит из двух слов 00 и 11 , то любое их соединение, во-первых, имеет четную длину и, во-вторых, нули и единицы расположены парами; например, слово 0110 не принадлежит множеству U^2 .

Слово α_1 называют **началом** слова α , если существует слово α_2 такое, что $\alpha = \alpha_1 \alpha_2$; при этом слово α_1 называют **собственным началом** слова α , если $\alpha_2 \neq \Lambda$. Слово α_2 называют **окончанием** слова α , если существует слово α_1 такое, что $\alpha = \alpha_1 \alpha_2$; при этом слово α_2 называют **собственным окончанием** слова α , если $\alpha_1 \neq \Lambda$. В частности, пустое слово является началом и окончанием любого слова α , причем собственным, если $\alpha \neq \Lambda$. Множество начал слов из множества U обозначим через $\bar{U}$, а множество окончаний – через $\bar{U}$.

2. Рассмотрим алфавит $B_{(r)} = \{0, 1, \dots, r-1\}$, где $r \geq 2$, и произвольное множество G . В частности, G может быть конечным алфавитом, множеством натуральных чисел, схем или формул определенного типа, множеством слов в некотором алфавите и т.п. Произвольное отображение множества G в множество слов в алфавите B называется **r -ичным кодированием множества G** . Мы будем рассматривать только случай $r = 2$, т.е. **двоичные кодирования**, и в связи с этим под $\log X$ будем понимать $\log_2 X$.

Приведем примеры кодирований.

1) Кодирование E – двоичная запись натуральных чисел. Числу $n = 0$ ставится в соответствие слово $e(0) = 0$, а числу $n > 0$ двоичное слово

$$e(n) = b_1^{(n)} b_2^{(n)} \dots b_{l(n)}^{(n)}$$

наименьшей длины, удовлетворяющее условию

$$\sum_{j=1}^{l(n)} b_j^{(n)} \cdot 2^{l(n)-j} = n.$$

Очевидно, что запись числа n начинается с цифры 1 , т.е. $b_1^{(n)} = 1$, а число знаков $l(n)$ должно удовлетворять двойному неравенству $2^{l(n)-1} \leq n < 2^{l(n)}$ и, следовательно, $l(n) = \lfloor \log n \rfloor + 1 = \lceil \log (n+1) \rceil$ (здесь $\lfloor X \rfloor$ обозначает целую часть числа X , а $\lceil X \rceil$ – наименьшее целое, не меньшее

Х). Кодирование E является взаимно однозначным: при $n_1 \neq n_2$ слова $e(n_1)$ и $e(n_2)$ различны.

2) Кодирование E_k первых 2^k натуральных чисел. Каждому числу n ($0 \leq n < 2^k$) ставится в соответствие слово $e_k(n) = 0^{k-l(n)}e(n)$. Слово $e_k(n)$ называется **двоичной записью числа n с помощью k цифр**. При кодировании E_k первые 2^k натуральных чисел отображаются на множество двоичных слов длины k . В таблице 3.1 приведены слова, соответствующие числам от 0 до 15 при кодированиях E и E_4 .

Упражнение. Составьте коды $e(25)$, $e_5(25)$, $e_6(25)$, $e(30)$, $e_6(30)$, $e_7(30)$.

3) Отображение, применяемое при написании цифр почтового индекса (рис. 3.1). Каждая десятичная цифра, выделенная отправителем из девяти отрезков шаблона, кодируется для автоматического распознавания словом длины 9 в алфавите B : символами 1 отмечаются номера использованных линий (например, цифре 2 соответствует слово 100100101; цифре 5 – слово 110010011).

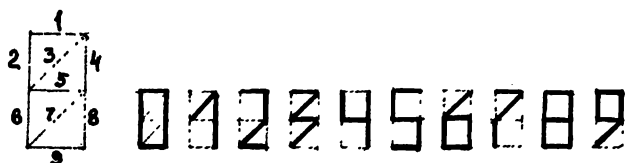


Рис. 3.1

Упражнение. Постройте код для 6-значного почтового индекса своего почтового отделения по месту жительства, используя описанные в п.2 двоичные 9-значные коды десятичных цифр.

Отображение конечных множеств на некоторые подмножества двоичных слов часто является удобным средством для подсчета или оценки числа элементов этих множеств. Рассмотрим, например, множество $\mathcal{F}_{n,m}$ упорядоченных разбиений числа n на m целых неотрицательных слагаемых. Каждому разбиению $n = n_1 + n_2 + \dots + n_m$ поставим в соответствие слово $0^{n_1}10^{n_2}1\dots10^{n_m}$. Оно имеет длину $(n + m - 1)$, т. к. содержит n нулей и $(m - 1)$ единиц, разделяющих серии нулей. Так, разбиение числа 11 на 4 слагаемых $11 = 5 + 0 + 2 + 4$ кодируется словом 00000110010000, разбиение $11 = 2 + 7 + 2 + 0$ – словом 001000000010011, а разбиение $11 = 1 + 1 + 6 + 3$ – словом 01010000001000. Такое кодирование является взаимно однозначным отображением множества $\mathcal{F}_{n,m}$ на множество двоичных слов длины $(n + m - 1)$, имеющих ровно $(m - 1)$ единиц. Следовательно, число элементов множества $\mathcal{F}_{n,m}$ равно C_{n+m-1}^{m-1} .

3. Особую роль в теории кодирования играет кодирование

Таблица 3.1

n	e(n)	e ₄ (n)	n	e(n)	e ₄ (n)	n	e(n)	e ₄ (n)
0	0	0000	6	110	0110	12	1100	1100
1	1	0001	7	111	0111	13	1101	1101
2	10	0010	8	1000	1000	14	1110	1110
3	11	0011	9	1001	1001	15	1111	1111
4	100	0100	10	1010	1010			
5	101	0101	11	1011	1011			

множества G , состоящего из всех (или выделенной части) слов в некотором алфавите A . Элементы кодируемого множества (т. е. выделенные слова в A) при этом называют **сообщениями**, имея в виду возможную передачу кода по каналу связи. В общем случае не накладывается никаких условий на процесс вычисления (сам алгоритм, его эффективность) для сообщения соответствующего ему двоичного слова. Однако для многих вопросов достаточно ограничиться рассмотрением более узких классов кодирований слов, так называемых побуквенных кодирований.

Пусть $A = \{a_i\}$ – конечный алфавит, буквы которого занумерованы натуральными числами. В этом случае кодирование букв алфавита A можно задать двоичными словами $V = \{v_i\}$, где v_i есть образ буквы a_i . Слова $V = \{v_i\}$ будем называть **кодами** (алфавита A). Кодирование слов в алфавите A , при котором каждому слову (сообщению) $a_{i_1} a_{i_2} \dots a_{i_k}$ ставится в соответствие слово $v_{i_1} v_{i_2} \dots v_{i_k}$, будем называть

побуквенным кодированием и обозначать через $K_{v_i}^{a_i}$ (или K_v^A).

Пример. Пусть $A = \{a, b, c, d\}$, двоичное кодирование его букв –
 $a \rightarrow 01, b \rightarrow 100, c \rightarrow 101, d \rightarrow 0$.

Слово 0100 можно декодировать как db или add; слово 0010100 – как ddcdd, daadd или dadb.

Заметим, что хорошо известный телеграфный код Морзе – кодирование букв алфавита (латинского или русского), цифр и знаков препинания словами в двухбуквенном алфавите “точка”, “тире” – не является двоичным, поскольку коды Морзе отделены дополнительным символом – разделителем: при передаче по каналам связи – временным (паузой), а на письме – пробелом (см. ниже §2, п. 2).

При переходе от взаимно однозначного кодирования букв алфавита к побуквенному кодированию слов свойство взаимной однозначности может не сохраниться. Например, при побуквенном кодировании $K_v^n(n)$

натуральных чисел кортежам из натуральных чисел (5, 6, 1), (11, 2, 1) и (5, 13) соответствует одно и то же слово 1011101 ($101 \cdot 110 \cdot 1$, $1011 \cdot 10 \cdot 1$, $101 \cdot 1101$). Таким образом, указанное побуквенное кодирование натуральных чисел не является взаимно однозначным.

Если коды всех букв алфавита имеют одинаковую длину, то код называется равномерным (пример – код E_n в отличие от кода E).

4. Можно считать, что в коде (множестве слов) $V = \{v_i\}$, содержатся только различные слова. Назовем код $V = \{v_i\}$ **разделимым**, если из

каждого равенства в алфавите $B = \{0, 1\}$ вида $v_{i_1} v_{i_2} \dots v_{i_k} = v_{j_1} v_{j_2} \dots v_{j_l}$ следует, что $k = l$ и $i_t = j_t$, $t = 1, 2, \dots, k$.

Очевидно, что побуквенное кодирование является взаимно однозначным тогда и только тогда, когда оно задается с помощью разделимого кода. Из определения следует также, что все слова разделимого кода различны и непусты.

Код $V = \{v_i\}$ будем называть **префиксным**, если никакое слово v_k не является началом никакого слова v_j , $j \neq k$. Ясно, что префиксный код является разделимым, т.е. *префиксность является достаточным условием разделимости*. Заметим, что если каждое слово префиксного кода заменить наименьшим его началом, которое не является началом других кодовых слов, то полученный код также будет префиксным. Такую операцию будем называть **усечением** префиксного кода.

Простейшим примером префиксного кода является равномерный код. Он не требует разделителей, поскольку код каждой передаваемой буквы заканчивается, а код следующей буквы начинается через определенное одинаковое число кодирующих символов.

Пример. Для 7-значных номеров московских телефонов префиксность (и, значит, разделимость) обеспечивается равномерностью. Для номеров же, начинающихся с 0: 01, 02, 03 и др., требование разделимости не допускает других номеров с тем же началом. Тем самым, наличие, например, кода 01 исключает $10^5 = 100000$ возможных 7-значных номеров.

Префиксности кода можно добиться и другим способом, использованным в упомянутом коде Морзе: включением в качестве окончания любого кодового слова специального символа – разделителя. Аналогично, ввод в компьютер имени файла, имени данного, многозначного числа и т.п. должен завершаться разделителем, в частности, нажатием клавиши <ENTER>.

Префиксность не является необходимым условием разделимости кода. Так, код $\langle a: 0, b: 01 \rangle$ двухбуквенного алфавита $\{a, b\}$ – не префиксный, но является разделимым. В самом деле, если в некотором сообщении после символа 0 следует 0, то первый 0 кодирует букву a,

так как 00 не является началом никакого кодового слова; если же после 0 следует 1, то пара символов 01 – код буквы b: 0 не может кодировать а, так как последующая единица не является началом никакого кодового слова.

Для произвольного кода V , состоящего из различных слов, рассмотрим ориентированный граф с множеством вершин $\bar{V}$, в котором вершина $\beta \in \bar{V}$ соединена с вершиной $\beta' \in \bar{V}$ дугой (β, β') тогда и только тогда, когда $\beta' = \beta 0$ или $\beta' = \beta 1$. Этот граф не содержит циклов и называется кодовым деревом кода V . Для префиксных кодов (и только для них) множество кодовых слов совпадает с множеством концевых вершин кодового дерева (т.е. вершин, из которых не исходит ребер). На рис. 3.2 изображено кодовое дерево кода, приведенного в таблице 3.2 ($l_i = \lambda(v_i)$).

Таблица 3.2

i	l_i	v_i
0	3	000
1	3	001
2	3	010
3	4	0110
4	4	0111
5	4	1000
6	4	1001

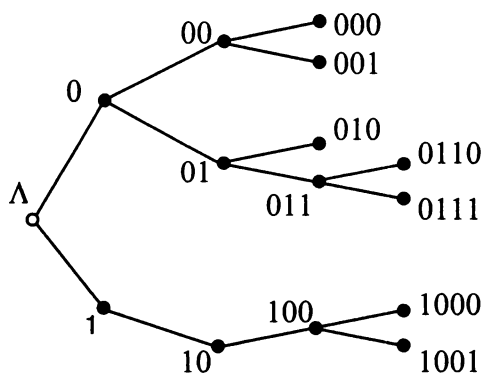


Рис. 3.2

Упражнение. Постройте кодовое дерево для кодов:

а) a = 000	б) a = 0000	в) a = 00
b = 010	b = 0001	b = 0101
c = 011	c = 0011	c = 011
d = 10	d = 01	d = 1100
e = 1100	e = 100	e = 1101
f = 1110	f = 101	f = 111
g = 1111	g = 111	

В дальнейшем для конечного кода $V = \{v_0, v_1, \dots, v_{m-1}\}$ будем считать, что $m \geq 2$, и пользоваться обозначением $\lambda_{\max} = \max_{0 \leq i < m-1} \lambda(v_i)$.

Теорема 1. Пусть $l_0, l_1, \dots, l_{m-1}$ – произвольный набор натуральных чисел ($m \geq 2$). Для того чтобы существовал разделимый код $V = \{v_0, v_1, \dots, v_{m-1}\}$ с длинами $\lambda(v_i) = l_i$, $i = 0, 1, \dots, m-1$, необходимо и достаточно выполнение **неравенства Крафта-Макмиллана для разделимых кодов**:

$$\sum_{i=0}^{m-1} \frac{1}{2^{l_i}} \leq 1.$$

Для кода табл. 3.2 выполнено: $3 \cdot \frac{1}{2^3} + 4 \cdot \frac{1}{2^4} = \frac{3}{8} + \frac{4}{16} = \frac{5}{8} < 1$.

Следствие. Для любого разделимого кода $V = \{v_0, v_1, \dots, v_{m-1}\}$ существует префиксный код с таким же набором длин кодовых слов, что и у кода V .

§ 2. Оптимальное кодирование

1. Здесь мы разберем построение эффективных взаимно однозначных кодирований при некоторых простейших предположениях относительно статистических свойств источника сообщений. При этом более эффективными считаются кодирования, у которых в среднем на одну букву сообщения приходится меньшее число двоичных цифр.

Рассматривается источник, который случайным образом (т.е. в случайном порядке) последовательно порождает буквы алфавита $A = \{a_0, a_1, \dots, a_{m-1}\}$. Предполагается, что последовательные появления букв алфавита A статистически независимы и подчинены распределению вероятностей

$$P = \{p_0, p_1, \dots, p_{m-1}\} \quad (p_i \geq 0, \sum_{i=0}^{m-1} p_i = 1).$$

Каждый код $V = \{v_0, v_1, \dots, v_{m-1}\}$ в алфавите $B = \{0, 1\}$ характеризуется средним числом

$$L_V(P) = \sum_{i=0}^{m-1} p_i \cdot \lambda(v_i)$$

двоичных цифр, приходящихся на одну букву алфавита A при побуквенном кодировании $K_{v_0}^{a_0}, K_{v_1}^{a_1}, \dots, K_{v_{m-1}}^{a_{m-1}}$. Величину $L_V(P)$ называют **стоимостью** кода V при распределении P . Пусть $L(P) = \inf L_V(P)$, где нижняя грань берется по (счетному) множеству префиксных кодов V , состоящих из m слов. Префиксный код $V = \{v_0, v_1, \dots, v_{m-1}\}$ будем называть **оптимальным** при распределении $P = \{p_0, p_1, \dots, p_{m-1}\}$, если $L_V(P) = L(P)$. Основная задача, которая здесь возникает, состоит в отыскании методов построения оптимальных или близких к ним по стоимости кодов и в оценке величины $L(P)$.

Известны два метода построения кодов, близких к оптимальному, принадлежащие К. Шеннону и Р. Фано.

Метод Фано, отличающийся чрезвычайной простотой конструкции, заключается в следующем. Упорядоченный (в порядке убывания вероятностей) список букв делится на две (последовательные) части так, чтобы суммы вероятностей входящих в них букв как можно меньше отличались друг от друга. Буквам из первой части приписывается символ 0, а буквам из второй части – символ 1. Далее точно так же поступают с каждой из полученных частей, если она содержит по крайней мере две буквы. Этот дихотомический процесс продолжается до тех пор, пока весь список не разобьется на части, содержащие по одной букве. Каждой букве ставится в соответствие последовательность символов, приписанных в результате этого процесса данной букве. Легко видеть, что полученный код является префиксным. Пример кода, построенного по методу Фано, приводится в таблице 3.3.

Таблица 3.3

сообщ.	p				код Фано		
a	0.30	} 0.48	0.30	0	00		
b	0.18		0.18	1	01		
c	0.14	} 0.52	} 0.28	0.14	0	100	
d	0.14			0.14	1	101	
e	0.11		} 0.24	} 0.13	0.11	0	110
f	0.06				0.06	0	1110
g	0.05	} 0.07	} 0.02	0.05	0	11110	
h	0.02			0.02	1	11111	

Стоимость кода:

$$2 \cdot (0.30 + 0.18) + 3 \cdot (0.14 + 0.14 + 0.11) + 4 \cdot 0.06 + 5 \cdot (0.05 + 0.02) = 2.72.$$

Следует заметить, что деление пополам списка букв может при определенном соотношении частот осуществляться неоднозначно.

Упражнение. Постройте два различных кода Фано для набора частот (0.4, 0.2, 0.15, 0.15, 0.1) и вычислите их стоимость.

2. В распространенном в телеграфии коде Морзе каждой букве или цифре, а также знакам препинания и некоторым другим символам сопоставляется некоторая последовательность кратковременных посылок тока ("точек") и в 3 раза более длинных посылок тока ("тире": 3 кратких посылки с малыми промежутками), разделяемых кратковременными паузами той же длительности, что и "точки". Пробел между буквами изображается разделителем-паузой (выключением тока на 3 единицы времени), длительность которой равна длительности "тире", а пробел между словами – двойной паузой.

Код Морзе создан с учетом частоты употребления разных букв

латинского алфавита в европейских языках. В таблице 3.4 приведены коды латинских букв и их частоты в англоязычных текстах. Более употребительные буквы, в основном, кодируются более короткими словами. Если кодировать английский текст по буквам, не учитывая указанных частот (и частоты пробела), то потребуется в среднем $\log_2 26 \approx 4.70$ двоичных знаков на одну букву (или $4.75 \approx \log_2 27$ – с учетом пробела). При использовании кода Морзе – 4.14 знака (экономия около 13%).

Таблица 3.4

.	E	0.13105	---	D	0.03788	---	W	0.01539
-	T	0.10468	----	L	0.03389	----	B	0.01440
..	A	0.08151	----	F	0.02924	----	V	0.00919
---	O	0.07995	----	C	0.02758	---	K	0.00420
..	N	0.07098	--	M	0.02536	----	X	0.00166
---	R	0.06882	---	U	0.02459	----	J	0.00132
..	I	0.06345	---	G	0.01994	----	Q	0.00121
...	S	0.06101	----	Y	0.01982	----	Z	0.00077
----	H	0.05259	----	P	0.01982			

Код Морзе для кириллицы составлен с учетом совпадения с латинским оригиналом для букв, изображающих одинаковые звуки (ср. русские Б и Н с латинскими В и N). Поэтому коды не соответствуют частотам встречаемости букв в русскоязычных текстах. В таблице 3.5 приведены коды и соответствующие частоты (буквы Ъ и Ь, а также Е и Ё кодируются одинаково).

Таблица 3.5

----	О	0.110	--	М	0.031	----	Ч	0.015
.	Е	0.087	---	Д	0.030	----	Й	0.012
..	А	0.075	----	П	0.028	----	Х	0.011
..	И	0.075	---	У	0.025	----	Ж	0.009
-	Т	0.065	----	Я	0.022	----	Ю	0.007
..	Н	0.065	----	Ы	0.019	----	Ш	0.007
...	С	0.055	----	З	0.018	----	Ц	0.005
---	Р	0.048	----	Ь	0.017	----	Щ	0.004
---	В	0.046	----	Ъ	0.017	----	Э	0.003
----	Л	0.042	----	Б	0.017	----	Ф	0.002
---	К	0.034	---	Г	0.016			

Без учета частот различных букв для 32-буквенного алфавита требуются в среднем $\log_2 32 = 5.00$ двоичных знаков на одну букву (для 31 букв – $\log_2 31 \approx 4.95$). С учетом реальных частот – 4.45 (экономия около 10%), а при использовании кода Морзе – несколько больше. Интересно, что аналогичный подсчет для частот букв в тексте романа Л. Толстого “Война и мир” с учетом пробелов между словами дает более низкое значение: 4.35.

Стоит заметить, однако, что неравномерный код Морзе с разделителем был позднее заменен в телеграфии равномерным 5-значным двоичным кодом Бодо (со сменой регистра, когда одно и то же 5-значное число кодирует два или три разных символа) из-за малой пригодности для буквопечатающего приема. В коде Бодо не требуется разделителя, т.к. на приемном конце канала известно, что через каждые 5 элементарных сигналов кончается одна буква и начинается следующая. Применяются также 6-значные коды без смены регистра.

§ 3. Коды с обнаружением и исправлением ошибок

1. Рассмотрим множество B^n слов длины n в алфавите $B = \{0, 1\}$. Каждое слово $x_1 x_2 \dots x_n$ в алфавите B отождествим с n -мерным вектором $(x_1, x_2, \dots, x_n)$. Суммой векторов $(x_1, x_2, \dots, x_n)$ и $(y_1, y_2, \dots, y_n)$ из B^n назовем вектор $(x_1 \oplus y_1, x_2 \oplus y_2, \dots, x_n \oplus y_n)$. Напомним, что знак $\oplus$ означает сумму по модулю 2. Если ввести еще умножение произвольного вектора $X = (x_1, x_2, \dots, x_n)$ на элемент $b \in B$ следующим образом: $bX = (bx_1, bx_2, \dots, bx_n)$, то множество B^n можно рассматривать как n -мерное векторное пространство.

В векторном пространстве B^n можно определить **расстояние Хемминга** $d(X, Y)$ между векторами X и Y как число несовпадающих компонент векторов X и Y и **норму** $\|X\|$ вектора X как расстояние между X и нулевым n -мерным вектором $(0, 0, \dots, 0)$. Очевидно, что для любого слова $X = x_1 x_2 \dots x_n$ из B^n :

$$\|X\| = \sum_{j=1}^n x_j$$

и для любых X и Y из B^n :

$$d(X, Y) = \|X \oplus Y\|$$

Метрическое пространство B^n допускает уже рассматривавшуюся нами простую геометрическую интерпретацию: множеству B^n соответствует множество вершин n -мерного единичного куба, а расстояние между

двумя элементами из B^n равно минимальному числу ребер в цепи, соединяющей соответствующие вершины куба.

Для произвольного слова $X = x_1x_2\dots x_n \in B^n$ определим его *числовое значение* $N(X)$:

$$N(X) = \sum_{j=1}^n x_j \cdot 2^{n-j}.$$

Примеры. Пусть $X = 0110011$ и $Y = 0001101$. Тогда $\|X\| = 4$, $\|Y\| = 3$,

$X \oplus Y = 0111110$, $d(X, Y) = \|X \oplus Y\| = 5$, $N(X) = 1 + 2 + 16 + 32 = 51$, $N(Y) = 1 + 4 + 8 = 13$. Заметим также, что, например, для числа 19 $e(19) = 10011$, $e_7(19) = 0010011$.

В дальнейшем в этом разделе будут рассматриваться лишь коды, состоящие из $m \geq 2$ двоичных слов фиксированной длины, причем в тех случаях, когда нумерация слов кода не существенна, мы не будем ее указывать. Для произвольного кода $V = \{v_0, v_1, \dots, v_{m-1}\} \subseteq B^n$ положим:

$$d(V) = \min_{i \neq j} d(v_i, v_j).$$

Величина $d(V)$ называется **кодovým расстоянием**.

Упражнение. Для кода $V = \{a_1: 0100, a_2: 1101, a_3: 1110, a_4: 0010\}$ вычислить попарные расстояния $d(a_i, a_j)$, $i \neq j$ и кодовое расстояние $d(V)$.

2. Рассмотрим некоторые виды преобразований двоичных слов, называемых **ошибками**.

Одиночной ошибкой вида $0 \rightarrow 1$ ($1 \rightarrow 0$) в слове X называют результат замены одного из символов 0 (соответственно 1) символом 1 (соответственно 0). Одиночные ошибки этого вида называют также **замещениями** символов, или **аддитивными ошибками**.

Одиночной ошибкой вида $0 \rightarrow \Lambda$ ($1 \rightarrow \Lambda$) в слове X называют результат удаления одного из символов 0 (соответственно 1); при этом длина слова уменьшается на единицу. Одиночные ошибки этого вида называются **выпадениями символов**.

Одиночной ошибкой вида $\Lambda \rightarrow 0$ ($\Lambda \rightarrow 1$) называют результат *вставки символов* перед некоторым символом слова или после его последнего символа; при этом длина слова увеличивается на единицу.

Одиночной ошибкой вида $+2^i$ (-2^i) в слове $X \in B^n$, где $0 \leq i < n$, называют преобразование слова X в слово $Y \in B^n$, числовое значение

которого на 2^i больше (соответственно меньше) числового значения слова X . Одиночные ошибки вида $+2^i$ и -2^i называются **арифметическими ошибками**.

Для иллюстрации в таблице 3.6 приведены слова, полученные из слова 0001101 (двоичная запись числа 13) в результате ошибок рассматриваемых видов.

Таблица 3.6

Вид ошибки	Место ошибки	Слово, полученное в результате ошибки	Примечание
$0 \rightarrow 1$	2 символ	0101101	
$1 \rightarrow 0$	5 символ	0001001	
$0 \rightarrow \Lambda$	2 символ	001101	
$1 \rightarrow \Lambda$	5 символ	000101	
$\Lambda \rightarrow 0$	между 2 и 3 символами	00001101	
$\Lambda \rightarrow 1$	между 2 и 3 символами	00101101	
$+2^2$		0010001	$13 + 4 = 17$
-2^2		0001001	$13 - 4 = 9$

Типом ошибки будем называть некоторое множество видов одиночных ошибок. Например, ошибка типа $\{0 \rightarrow \Lambda, 1 \rightarrow \Lambda, \Lambda \rightarrow 0, \Lambda \rightarrow 1\}$ есть выпадение или вставка произвольного символа. Число одиночных ошибок в некоторой их последовательности будем называть **кратностью** ошибки. Так, в результате ошибки кратности 3 (вставка 1 перед первым символом, затем выпадение 0 перед пятым и замещение последнего символа) слово 0001101 переводится в слово 1001100.

В дальнейшем особое внимание будет уделено ошибкам типа $\{(0 \rightarrow 1), (1 \rightarrow 0)\}$, т.е. замещениям символов.

При постановке различных задач обнаружения и исправления ошибок заданного типа исходят из того или иного допущения о законе образования ошибок. Эти законы могут иметь как вероятностный, так и комбинаторный характер. Говорят, что тот или иной закон образования ошибок задается каналом передачи (или хранения) двоичных слов. Наиболее часто рассматриваются следующие два типа каналов:

а) в любом символе с вероятностью p ($0 < p < 1/2$) происходит ошибка заданного типа, причем ошибки в различных символах статистически независимы;

б) в каждом слове длины n может произойти любая ошибка заданного типа кратности не более s .

Основная идея помехоустойчивого кодирования состоит в следующем. Вместо сообщений $X, Y, Z, \dots$ по каналу связи посылают их определенным образом закодированные образы $X', Y', Z', \dots$, которые обеспечивают обнаружение ошибок или правильное декодирование даже при наличии ошибок допустимого типа.

Возможность исправления ошибок при использовании кода $V = \{v_0, v_1, \dots, v_n\} \subseteq B^n$ можно пояснить следующим образом. Обозначим через $T(X)$ множество слов, которые можно получить из слова X в результате ошибок, *допустимых в рассматриваемом канале*, в частности $T(v_i)$ – множество слов, в которые может превратиться в результате ошибок кодовое слово v_i . Произвольное однозначное отображение D множества

$\bigcup_{i=0}^{m-1} T(v_i)$ на V будем называть **декодированием**. Задание декоди-

рования D равносильно разбиению множества $\bigcup_{i=0}^{m-1} T(v_i)$ на непересекающиеся подмножества (**окрестности**) $D^{-1}(v_i)$, $i = 0, 1, \dots, m-1$, где каждая окрестность $D^{-1}(v_i)$ состоит из прообразов слова v_i при отображении D . Естественнo считать, что при декодировании D в произвольном слове $v_i \in V$ исправляются те и только те ошибки, которые преобразуют слово v_i в некоторое слово из $T(v_i) \cap D^{-1}(v_i)$. В частности, для существования декодирования D , при котором исправляются все ошибки, допустимые в рассматриваемом канале, необходимо и достаточно, чтобы множества $T(v_i)$, $i = 0, 1, \dots, m-1$, попарно не пересекались. Тогда, получая по каналу связи некоторое сообщение Y , мы определяем, окрестности какого кодового слова X оно принадлежит, и делаем вывод, что послано именно слово X . Код, обладающий таким свойством, называется кодом, исправляющим ошибки данного типа.

Для кода $V \subseteq B^n$ с исправлением s ошибок типа замещения множество $T(X)$ совпадает с *метрической окрестностью радиуса s слова X* , т.е. множеством точек, удаленных от X на расстояние не более s . Поэтому условие непересечения множеств $T(v_i)$ равносильно тому, что кодовое расстояние $d(V) > 2s$. Таким образом, код V является **кодом с исправлением s замещений** тогда и только тогда, когда $d(V) > 2s$, т.е. $d(V) \geq 2s + 1$, поскольку $d(V)$ – целое число.

Наряду с задачами исправления ошибок можно рассматривать и более слабые задачи обнаружения ошибок. Очевидно, при использовании кода V нельзя обнаружить лишь те ошибки, которые преобразуют кодовые слова в другие кодовые слова. Отсюда, в частности, следует, что код V с кодовым расстоянием d всегда позволяет обнаружить $(d - 1)$ или менее одиночных ошибок типа замещения.

3. Из сказанного выше можно заключить, что для обеспечения помехоустойчивости кодирования приходится применять более длинные коды, в которых кроме информации о самом сообщении содержится дополнительная информация, позволяющая при возникновении ошибок в процессе передачи (или хранения) сообщения обнаруживать или восстанавливать переданное сообщение. Этот факт можно сформулировать таким образом: за надежность приходится платить избыточностью кодирования.

Так, для кодирования 10 арабских цифр достаточно использовать не 9 двоичных знаков, как описано в примере 3 §1, а 4 ($2^4 = 16 > 10$), и соответственно минимизировать визуальный способ написания (пример на рис. 3.3). Однако при таком способе кодирования среди $C_{10}^2 = 45$ пар кодов имеется 14 пар, различающихся ровно в одном разряде, и даже при одиночном замещении можно принять один из кодов за другой. В реально используемом кодировании десятичных цифр 9-значными двоичными числами только в одной паре (цифры 0 и 8) различие в одном разряде (что, по-видимому, приводит иногда к ошибкам, связанным с неясным заполнением или сбоем при прочтении); в остальных парах – не менее, чем в трех разрядах. Кроме того, принятый метод кодирования предпочтителен по другой причине: он удобнее, поскольку имеет сходство с обычным начертанием арабских цифр.

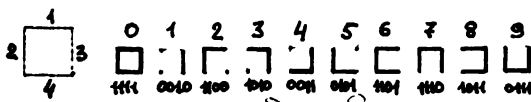


Рис. 3.3

Простейший способ построения помехоустойчивых кодов – дублирование. Так, если каждую букву передаваемого слова повторить $(s + 1)$ раз, то такой код способен обнаружить до s ошибок типа замещения: при отсутствии ошибок сообщение разбивается на однородные серии из $(s + 1)$ одинаковых символов; если же произошли ошибки, то, даже если все s ошибок – внутри одной серии, можно обнаружить символ или несколько символов, которые, возможно, переданы с ошибкой, – те, в которых нарушена однородность.

Пример. Слово 01101 для возможности обнаружения до 2 ошибок можно закодировать как 000111111000111. Тогда при получении по каналу связи сообщения

000010111000111 (*)

следует разделить его на тройки: 000 010 111 000 111, после чего

видно, что во втором символе – ошибка. Однако мы не можем восстановить передаваемое сообщение (т.е. достоверно утверждать, что посланное слово именно 01101 и произошла двойная ошибка во втором символе), поскольку такое же сообщение (*) могло быть получено из слова 00101 при одиночной ошибке во втором символе.

Аналогично, повторение каждого символа передаваемого слова $(2s + 1)$ раз позволяет не только обнаружить, но и исправить до s ошибок типа замещения. Для предыдущего примера слово 01101 должно быть передано как 0000011111111110000011111. При не более чем двух ошибках, даже если они внутри одной серии, большинство букв серии – не замещены и определяют правильное декодирование. Если, например, получено слово 0000001101111110000011111, то после разбиения его на пятерки символов 00000 01011 11111 00000 11111 видно, во-первых, что ошибка произошла во втором символе, и, во-вторых, что исходная буква – 1 (число замещенных символов меньше половины длины серии).

4. Далее мы будем рассматривать только случай одиночной ошибки типа замещения. При использовании описанных простейших приемов для возможности обнаружения одиночной ошибки длина передаваемого слова удваивается, а для ее исправления – утраивается.

Однако возможно значительно более экономное помехоустойчивое кодирование. Идея таких методов заключается в следующем. На множестве двоичных слов рассматривается некоторая функция. Искомый код определяется как множество слов из B^n , на которых эта функция принимает некоторое фиксированное значение. Функция подбирается таким образом, чтобы в результате любой одиночной ошибки значение функции изменялось и чтобы по этому изменению и, быть может, самому полученному слову можно было однозначно определить вид и место ошибки.

Мы рассмотрим один пример такого кодирования – код Хэмминга.

Зафиксируем число n и найдем число l такое, что $2^{l-1} \leq n < 2^l$. В этом случае $l = \lceil \log n \rceil + 1$. Например, $l(5) = l(7) = 3$, $l(8) = l(10) = l(13) = 4$.

Для произвольного слова $X = x_1x_2...x_n \in B^n$ положим

$$H(X) = x_1e_l(1) \oplus x_2e_l(2) \oplus \dots \oplus x_ne_l(n).$$

$H(X)$ представляет собой вектор длины l , полученный в результате сложения векторов, являющихся двоичными записями (с помощью l цифр) номеров единичных символов слова X .

Примеры. Пусть $n = 6$, $X = 010101$ и $Y = 110100$. Тогда $l = l(6) = 3$,

$$H(X) = (0, 1, 0) \oplus (1, 0, 0) \oplus (1, 1, 0) = (0, 0, 0),$$

$$H(Y) = (0, 0, 1) \oplus (0, 1, 0) \oplus (1, 0, 0) = (1, 1, 1).$$

Теорема. Код Хэмминга H_n , состоящий из всех слов $X = x_1x_2\dots x_n \in B^n$ таких, что

$$H(X) = (0, 0, \dots, 0), \quad (*)$$

является кодом с исправлением одного замещения.

В рассмотренном выше примере $X \in H_6$, $Y \notin H_6$.

Пример. Пусть в слове $X = 010101 \in H_6$ произошло замещение пятого символа, в результате чего получилось слово $T = 010111$. Так как $H(T) = (0, 1, 0) \oplus (1, 0, 0) \oplus (1, 0, 1) \oplus (1, 1, 0) = (1, 0, 1)$, то числовое значение слова $H(Y)$ равняется 5 и определяет номер замещенного символа.

Заметим, что $|H_n|$ – число элементов кода H_n – равно 2^{n-l} . Это следует из того, что при произвольном задании значений $(n - l)$ компонент, номера которых отличны от $2^0, 2^1, \dots, 2^{l-1}$ (**информационные позиции**), значения компонент с номерами $2^0, 2^1, \dots, 2^{l-1}$ (**проверочные позиции**) однозначно определяются из условия (*).

Так как $l = \lceil \log n \rceil + 1 = \lceil \log(n+1) \rceil$, то

$$\frac{2^{n-l}}{n} \leq |H| = 2^{n-\lceil \log(n+1) \rceil} \leq \frac{2^n}{n+1}.$$

В частности, $|H_6| = 2^{6-\lceil \log 7 \rceil} = 2^{6-3} = 2^3 = 8$. Код H_6 приведен ниже:

H_6
 000000
 111000
 110011
 001011
 101101
 010101
 011110
 100110

Упражнение. Проверьте, что любые два элемента кода H_6 различаются не менее, чем в трех разрядах.

Таким кодовым множеством можно кодировать все слова длины 3 – их тоже 8, сопоставив любым образом взаимно однозначно. Р.Хэммингом предложен способ кодирования, обеспечивающий простое и удобное декодирование. Для этого кодируемое слово X длины m дополняется l проверочными разрядами ($l = \lceil \log(m+1) \rceil$), которые

определенным образом рассчитываются при кодировании, и полученное сообщение X состоит из m информационных и l проверочных позиций. Для проверочных разрядов отводятся 1-й, 2-й, 4-й, 8-й и т.д., номера которых являются целыми степенями числа 2: их двоичные представления содержат ровно одну единицу. На остальные места: 3, 5, 6, 7, 9, 10,... помещают символы кодируемого слова X .

Покажем на двух примерах, как проводится кодирование по Хэммингу. Пусть кодируемое слово $X = 1100$: $m = 4$, $l = \lceil \log_2 5 \rceil = 3$. Закодированное сообщение длины $m + l = 7$ будет иметь вид $p_1 p_2 p_3 p_4 p_5 p_6 p_7$. Проверочные символы p_1, p_2, p_4 вычисляются следующим образом. p_i равно сумме по модулю 2 тех информационных символов, номера которых имеют единицу в двоичном представлении там же, где и номер p_i , т.е. в i -ом разряде справа: p_1 – в 1-ом разряде, p_2 – во 2-ом, p_4 – в 3-ем (таблица 3.7). $p_1 = p_3 \oplus p_5 = 0$; $p_2 = p_3 = 1$; $p_4 = p_5 = 1$. В результате получаем $X' = 0111100$.

Таблица 3.7

i	i в двоичн. представлении	Информационные позиции	Проверочные позиции	Расчет проверочных символов	Сообщение после кодирования
1	001		0	$p_3 \oplus p_5$	0
2	010		1	p_3	1
3	011	1			1
4	100		1	p_5	1
5	101	1			1
6	110	0			0
7	111	0			0

Пример. Кодируемое слово X длины $m = 10$: 1001110110; $l = 4$, вычисление проверочных разрядов – в таблице 3.8, $X' = 01110010110110$.

Таблица 3.8

i	i в двоичн. представ.	Информационные позиции	Проверочные позиции	Расчет проверочных символов
1	0001		0	$p_3 \oplus p_7 \oplus p_9 \oplus p_{13}$
2	0010		1	$p_3 \oplus p_7 \oplus p_{10}$
3	0011	1		
4	0100		1	$p_7 \oplus p_{12} \oplus p_{13}$
5	0101	0		

i	i в двоичн. представ.	Информ. позиции	Проверочн. позиции	Расчет проверочных символов
6	0110	0		
7	0111	1		
8	1000		0	$p_9 \oplus p_{10} \oplus p_{12} \oplus p_{13}$
9	1001	1		
10	1010	1		
11	1011	0		
12	1100	1		
13	1101	1		
14	1110	0		

Докажем теперь, что для любых двух различных слов X, Y их закодированные по Хэммингу образы X', Y' различаются не менее чем в трех разрядах, что и обеспечивает исправление единичной ошибки.

1) $\rho(X, Y) \geq 3$, т.е. X, Y различаются в трех или больше разрядах: при кодировании эти различия сохраняются; могут добавиться различия в проверочных разрядах, т.е. $\rho(X', Y') \geq 3$.

2) $\rho(X, Y) = 2$, т.е. X, Y различаются в двух разрядах. Пусть в закодированных словах это позиции $\alpha \neq \beta$, числа α, β в двоичной записи различаются хотя бы в одном (i -ом) разряде, который для α равен 0, а для β равен 1 или наоборот. При вычислении проверочного символа, соответствующего i -му разряду, т.е. позиции 2^i , только для одного из слов X', Y' в сумме по модулю 2 участвует единица; по остальным позициям, отличным от α, β , слова X и Y совпадают. Значит, в 2^i -ой проверочной позиции слова X' и Y' также различаются: в итоге $\rho(X', Y') \geq 3$.

3) $\rho(X, Y) = 1$, т.е. X, Y различаются ровно в одном разряде. Номер этого информационного разряда в словах X и Y не равен целой степени числа 2, т.е. его двоичная запись содержит по крайней мере две единицы. Тогда в соответствующих двух проверочных разрядах слова X и Y различаются: снова $\rho(X', Y') \geq 3$.

Если при передаче кодированного слова X произошла одна ошибка, то при декодировании полученного слова T вектор $H(T)$, прочитанный как двоичная запись, дает номер поврежденного разряда.

Упражнение. В словах X' двух последних примеров измените один символ, вычислите $H(T)$ и убедитесь в справедливости вышесказанного.

ГЛАВА IV. ЛОГИЧЕСКИЕ СЕТИ И АВТОМАТЫ

§1. Схемы из функциональных элементов для систем булевых функций

1. С определением и примерами **схем из функциональных элементов** мы уже встречались. Используя терминологию предыдущей главы, можно считать схему многополюсной сетью, входы схемы - входными полюсами, а выход - выходным полюсом. Элементы являются вершинами схемы, а их соединения - дугами, исходящими из входных полюсов или выходов других элементов и входящими во входы элементов или в выходной полюс.

Здесь мы будем рассматривать схемы из элементов, реализующих булевы функции. Для построения (синтеза) такой схемы функция должна быть представлена формулой, выражающей эту функцию через базисные функции, которые реализуются базисными элементами; они могут быть различными и должны обеспечивать возможность выразить их суперпозициями функции требуемого класса. Например, любую функцию можно выразить в виде ДНФ.

Тогда достаточно иметь набор элементов, содержащий:

конъюнктор - элемент с двумя входами, реализующий конъюнкцию;

дизъюнктор - элемент с двумя входами, реализующий дизъюнкцию;

инвертор - элемент с одним входом, реализующий отрицание.

В некоторых случаях удобно использовать и другие элементы, например, двухвходовый **элементарный сумматор**, реализующий сумму по модулю 2.

Расширим понятие схемы из функциональных элементов, допуская несколько различных выходов схемы, которая в этом случае реализует не одну функцию, а несколько (рис. 4.1а). Конечно, можно реализовать систему различных функций $f_1, f_2, \dots, f_k$, построив для них отдельные

схемы $\sum_1, \sum_2, \dots, \sum_k$ совмещая (склеивая) соответствующие входные полюсы (рис. 4.1б). Однако некоторые внутренние части различных схем могут совпадать, и это можно использовать, чтобы не дублировать их и тем самым сократить общее число элементов схемы. Пример такого рода уже встречался ранее [юнита 1, гл. I] при реализации среднего значения Mx и выборочной дисперсии Sx для статистической выборки.

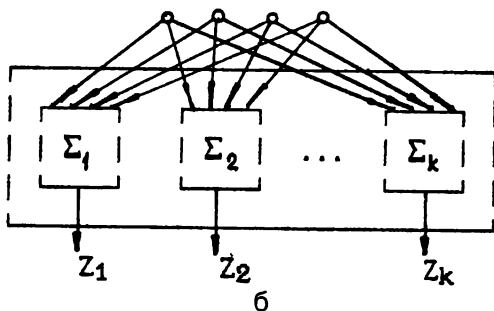
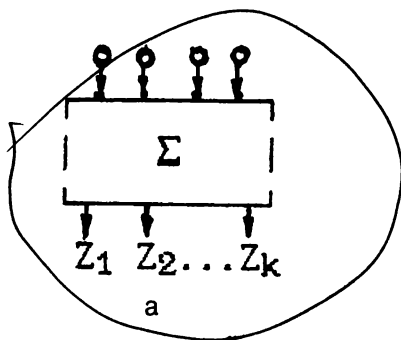


Рис. 4.1

2. Рассматривая арифметические действия над двоичными числами (сложение, умножение и др.), мы отмечали, что знаки результата являются булевыми функциями от булевых переменных - знаков (цифр) слагаемых/сомножителей. Именно: если X и Y - однозначные числа, то арифметическая сумма $X + Y$ представляет собой, вообще говоря, двузначное число $\tilde{Z} = Z_1 Z_0$, знаки которого выражаются так:

$$Z_0 = X \oplus Y, \quad Z_1 = X \& Y,$$

где Z_1 - знак переноса в старший разряд - равняется 1, если оба слагаемых равны 1 ($1 + 1 = 10$).

Соответствующая схема из функциональных элементов изображена на рис. 4.2.

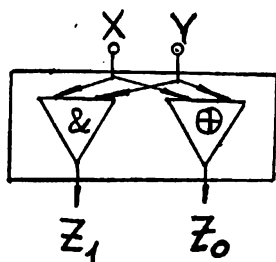


Рис. 4.2

При сложении двух двузначных, трехзначных и т.д. чисел младший разряд суммы определяется также: $Z_0 = X_0 \oplus Y_0$. Но возникающий знак переноса складывается с двумя знаками предыдущего (старшего) разряда; знак переноса из второго разряда в третий складывается, в свою очередь, с третьими справа знаками слагаемых и т.д.

Упражнение. Разберите подробно, как производятся поразрядные операции при сложении двоичных чисел 1011 и 1101: как определяются двоичные знаки суммы и знаки переноса.

Таким образом, при сложении многозначных чисел $\tilde{X} = X_{n-1} X_{n-2} \dots X_1 X_0$ и $\tilde{Y} = Y_{n-1} Y_{n-2} \dots Y_1 Y_0$ $(k+1)$ -й справа знак Z_k суммы $\tilde{Z} = Z_n Z_{n-1} \dots Z_k \dots Z_1 Z_0$ представляет собой сумму по модулю 2 трех булевых слагаемых X_k , Y_k и σ_k , где σ_k - знак переноса из $(k-1)$ -го разряда в k -й:

$$Z_k = X_k \oplus Y_k \oplus \sigma_k.$$

Знак переноса из k -го разряда в $(k+1)$ -й равняется 1, если арифметическая сумма чисел $X_k + Y_k + \sigma_k$ превышает 1, т.е. если хотя бы два из трех слагаемых X_k, Y_k, σ_k равны 1. Это означает, что знак переноса равен значению функции большинства $m_3(X_k, Y_k, \sigma_k)$, которую можно выразить в виде ДНФ: $\sigma_{k+1} = m_3(X_k, Y_k, \sigma_k) = X_k \cdot Y_k \vee X_k \cdot \sigma_k \vee Y_k \cdot \sigma_k$.

Вычисление булевых переменных Z_k и σ_{k+1} можно реализовать схемами, содержащими 2 элементарных сумматора для Z_k и 2 дизъюнктора и 3 конъюнктора для σ_{k+1} (рис. 4.3а,б).

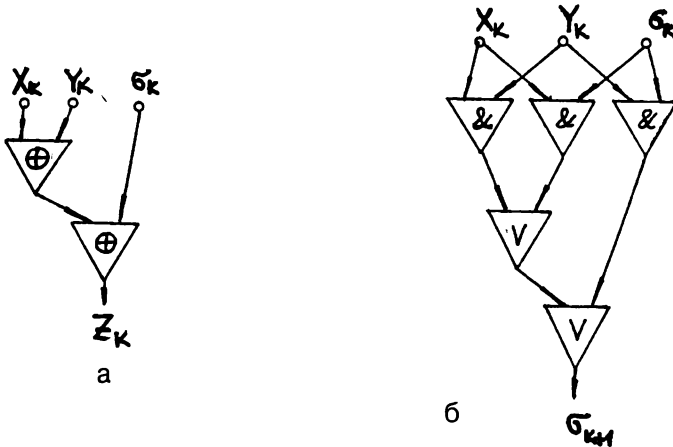


Рис. 4.3

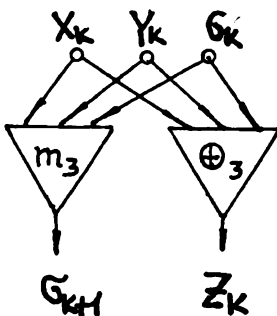


Рис. 4.4

Однако нам будет удобно использовать сокращенную форму (рис. 4.4), обозначая через $\oplus_3$ каскад из двух элементарных сумматоров для суммирования трех слагаемых (рис. 4.3а) и через m_3 - трехходовый элемент, реализующий функцию большинства (рис. 4.3б).

Склеивая входы двух этих элементов, мы получаем подсхему S, изображенную на рис. 4.5, которая представляет собой ячейку с тремя входами и двумя выходами, из нескольких экземпляров которой строится схема для реализации арифметического сумматора n-значных двоичных чисел $\tilde{X} = X_{n-1} X_{n-2} \dots X_1 X_0$ и $\tilde{Y} = Y_{n-1} Y_{n-2} \dots Y_1 Y_0$ (если числа $\tilde{X}$ и $\tilde{Y}$ имеют различную разрядность, то нужно дополнить меньшее нулями слева).

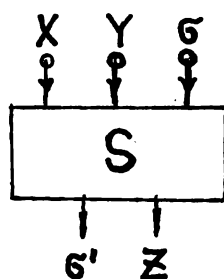


Рис. 4.5

Схему рис. 4.2 для расчета Z_0 и σ_0 можно (хотя и не обязательно) представить для единообразия как ту же ячейку S, если считать, что $\sigma_0 = 0$. Тогда общая схема для суммирования многозначных чисел может быть построена так, как на рис. 4.6. Схема имеет $2n$ входов и $(n + 1)$ выходов. Нетрудно подсчитать, что для сложения n-значных чисел требуется $8n$ (если для расчета Z_0 использовать более экономную подсхему рис. 4.2, то $8n - 6$) элементов; например, для 20-значных чисел - 154 элемента.

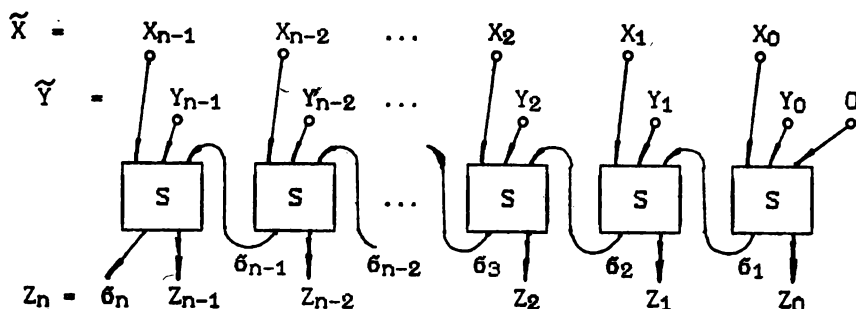


Рис. 4.6

Аналогично может быть построена и схема для умножения "столбиком" n-значных двоичных чисел $X = X_{n-1} \dots X_1 X_0$ и $Y = Y_{n-1} \dots Y_1 Y_0$. Рассмотрим в качестве примера случай $n = 3$, т.е. перемножение двух

4-значных чисел. В строках, обозначенных T_0, T_1, T_2, T_3 , записываются (со сдвигом) произведения числа X на однозначные числа 0 или 1, определяемые знаками числа Y , т.е. либо число X , либо 0. Затем записанные числа складываются; при этом число однозначных слагаемых в каждом разряде - от 1 до n . В общем случае k -й разряд произведения Z_k ($k = 0, 1, \dots, n-1$) зависит от знаков младших разрядов сомножителей, а при $k > n$ - от всех $2n$ знаков, но, в отличие от сложения, в определении Z_k участвуют несколько знаков переноса из младших разрядов. Результат перемножения двух n -значных чисел представляет $2n$ -значное число.

Пример для $n = 4$:

			$\tilde{X} = X_3$ $\tilde{Y} = Y_3$	X_2 Y_2	X_1 Y_1	X_0 Y_0			
$T_0 = \tilde{X} \cdot Y_0$, сдвиг 0				$X_3 \cdot Y_0$	$X_2 \cdot Y_0$	$X_1 \cdot Y_0$	$X_0 \cdot Y_0$		
$T_1 = \tilde{X} \cdot Y_1$, сдвиг 1				$X_3 \cdot Y_1$	$X_2 \cdot Y_1$	$X_1 \cdot Y_1$	$X_0 \cdot Y_1$		
$T_2 = \tilde{X} \cdot Y_2$, сдвиг 2			$X_3 \cdot Y_2$	$X_2 \cdot Y_2$	$X_1 \cdot Y_2$	$X_0 \cdot Y_2$			
$T_3 = \tilde{X} \cdot Y_3$, сдвиг 3		$X_3 \cdot Y_3$	$X_2 \cdot Y_3$	$X_1 \cdot Y_3$	$X_0 \cdot Y_3$				
.....									
		Z_7	Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0

$$\begin{array}{r}
 \begin{array}{r}
 1101 \\
 \times 1011 \\
 \hline
 1101 \\
 1101 \\
 1101 \\
 1101 \\
 \hline
 10001111
 \end{array}
 \quad = \quad
 \begin{array}{r}
 13 \\
 11 \\
 \\
 \\
 \\
 143
 \end{array}
 \end{array}$$

3. Свойства схем из функциональных элементов.

1) Отсутствие ориентированных циклов (контуров). Этим обеспечивается невозможность попадания выходного значения некоторого элемента на вход того же элемента, даже если оно преобразовано действием других элементов. Иначе могли бы возникнуть противоречия, когда, например, нулевое значение на входе при прохождении по контуру могло превратиться в единицу, и не понятно, какое именно значение должно поступить на вход.

2) Вычисление линейно упорядоченной системы функций можно рассматривать как вычисление - для каждого набора входных значений - многозначного двоичного числа, знаками которого является кортеж

выходных значений (слово в алфавите $B = \{0, 1\}$).

3) Следует считать, что вычисление совершается без учета времени, т.е. все вычисляется как бы одновременно и не возникает коллизий, что из нескольких входных аргументов какого-то элемента одни уже вычислены, а другие еще нет.

Пример. Пусть в подъезде 5-этажного дома работают 2 лифта. Требуется построить схему, которая, если оба лифта находятся выше 1-го этажа, определяла бы приоритет вызова одного из лифтов, так чтобы нажатием одной кнопки вызова можно было вызвать ближайший из них.

Занумеруем лифты числами 0 и 1, а этажи с 2 по 5 - числами 0, 1, 2, 3, которые будем представлять их двоичными записями 00, 01, 10, 11. Тогда совместное расположение обоих лифтов задается упорядоченной парой чисел $\alpha, \beta \in [0, 3] \subseteq N$, или четверкой булевых параметров $\alpha_1, \alpha_2, \beta_1, \beta_2$. Например, набор 0110 означает, что первый лифт находится на 3 этаже ($\alpha = 01$), а второй - на 4-м ($\beta = 10$).

Для построения схемы нужна формула, выражающая зависимость номера Z приоритетного лифта от переменных $\alpha_1, \alpha_2, \beta_1, \beta_2$. Составим таблицу 4.1.

Здесь Z принимает значение 0, если 1-й лифт находится ниже 2-го, и 1, если 2-й лифт - ниже 1-го. Значения представлены в графе Z . В случаях, когда оба лифта находятся на одном этаже, можно определить приоритет и, тем самым, функцию Z как угодно. Доопределим Z так, чтобы формула была возможно проще. Это можно сделать, как показано в столбце "доопределение Z ". В последнем столбце таблицы представлено разложение функции Z по двум первым переменным: $Z = \bar{\alpha}_1 \bar{\alpha}_2 \cdot 0 \vee \bar{\alpha}_1 \alpha_2 \cdot \bar{\beta}_1 \vee \alpha_1 \bar{\alpha}_2 \cdot \bar{\beta}_1 \vee \alpha_1 \alpha_2 \cdot 1$. Первое слагаемое в этой дизъюнкции четырех конъюнкций тождественно равно 0; из двух следующих можно вынести за скобки общий множитель $\bar{\beta}_1$: дизъюнкция $\bar{\alpha}_1 \alpha_2 \vee \alpha_1 \bar{\alpha}_2$ есть СДНФ функции $(\alpha_1 \oplus \alpha_2)$; четвертое слагаемое равно $\alpha_1 \alpha_2$. Окончательно получаем:

$$Z = \bar{\beta}_1 \cdot (\alpha_1 \oplus \alpha_2) \vee \alpha_1 \alpha_2.$$

Интересно, что при таком доопределении функция Z не зависит от переменной β_2 . Это значит, что на нахождение 2-го лифта на 2-м или на 3-м этаже система реагирует одинаково, так же как не различается

его нахождение на 4-м этаже от 5-го.

Соответствующая схема из 5 элементов изображена на рис. 4.7.

Таблица 4.1

α	β	α_1	α_2	β_1	β_2	Z Доопределение Z	Z после доопределения	$f_{\alpha, \alpha_2}(\beta_1, \beta_2)$
0	0	0	0	0	0	0	0	}
0	1	0	0	0	1	0	0	
0	2	0	0	1	0	0	0	
0	3	0	0	1	1	0	0	

1	0	0	1	0	0	1	1	}
1	1	0	1	0	1	1	1	
1	2	0	1	1	0	0	0	
1	3	0	1	1	1	0	0	

2	0	1	0	0	0	1	1	}
2	1	1	0	0	1	1	1	
2	2	1	0	1	0	0	0	
2	3	1	0	1	1	0	0	

3	0	1	1	0	0	1	1	}
3	1	1	1	0	1	1	1	
3	2	1	1	1	0	1	1	
3	3	1	1	1	1	1	1	

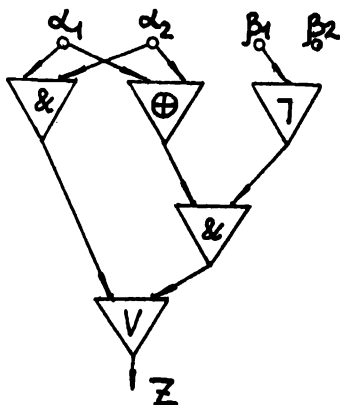


Рис. 4.7

§2. Логические сети

1. Расширим понятие схемы из функциональных элементов. Определим ее функционирование в дискретном времени $t = 0, 1, 2, \dots$. Считается, что в каждый момент времени на входы схемы подаются значения аргументов: для n -входовой схемы

$$X(t) = (x_1(t), x_2(t), \dots, x_n(t)).$$

Если схема состоит, как мы до сих пор рассматривали, только из функциональных элементов, то в тот же момент времени t на выходах схемы реализуются функции алгебры логики на входном наборе, определяемые структурой схемы и составом элементов, т.е. возникает выходной набор

$$Z(t) = (z_1(t), z_2(t), \dots, z_m(t)),$$

причем значения $z_i(t)$ зависят только от совокупности значений аргументов $x_1(t), x_2(t), \dots, x_n(t)$ в тот же момент времени.

Введем теперь новый тип элемента с одним входом, который называется **задержкой** и обозначается символом “з”; его функционирование определяется так: если $x(t)$ - входной аргумент элемента задержки, то $z(t+1) = x(t)$, т.е. значение на выходе задержки в каждый момент времени равно значению на входе элемента в предыдущий момент времени. Заметим, что значение $z(0)$ не может быть определено указанным равенством: полагают по определению, что $z(0) = 0$.

Включение элементов задержки в схему позволяет связывать значения, подаваемые на входы и вычисляемые внутри схемы (как выходные, так и промежуточные) в разные моменты времени.

В.95 Схема из функциональных элементов и задержек, действующая в тактовом (дискретном) режиме, называется **логической сетью**. При этом допустимы ориентированные циклы (контуры), если в них присутствует хотя бы один элемент задержки.

Существенное свойство логической сети: если задать входную последовательность $X(0), X(1), \dots$, то однозначно определяется выходная последовательность $Z(0), Z(1), \dots$. Действительно, на вход каждого функционального элемента или задержки подается либо входное значение, либо выход другого элемента, в частности, задержки. Поэтому для всех t определены выходные значения каждого элемента, а для $t = 0$ выходы всех задержек равны 0 по определению.

Пример 1 (рис. 4.8). Выходное значение $Z(t)$ определяется равенством $Z(t) = (X(t) \oplus Y(t)) \& z(t) = [\text{поскольку } z(t) = Y(t-1)] = (X(t) \oplus Y(t)) \& Y(t-1)$. $z(0) = 0$. Поэтому входная последовательность $(X(t); Y(t))$ перерабатывается схемой рис. 4.8 в выходную последовательность $Z(t)$ (таблица 4.2).

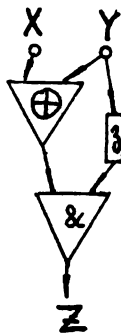


Рис. 4.8

Таблица 4.2

t	0	1	2	3	4	5	...
X	0	1	1	1	0	0	...
Y	1	0	1	0	1	1	...
Z	0	1	0	1	0	1	...

В этой логической сети нет контуров.

Пример 2 (рис. 4.9). $Z(t) = (X(t) \vee z(t) = (X(t) \vee Z(t-1))$.

Аналогично, $Z(t-1) = (X(t-1) \vee Z(t-2))$. Продолжая эту рекурсию,

получаем: $Z(t) = X(t) \vee X(t-1) \vee X(t-2) \vee \dots \vee X(0) = \bigvee_{\tau=0}^t X(\tau)$.

Исходя из определения дизъюнкции $Z(t) = \begin{cases} 0, & \text{если } \forall_{\tau \leq t} : X(\tau) = 0, \\ 1, & \text{в противном случае,} \\ & \text{т.е. если } \exists_{\tau \leq t} : X(\tau) = 1. \end{cases}$

Таким образом, $Z(t)$ тождественно равно 0 до появления на входе первой единицы, после чего тождественно равно 1.

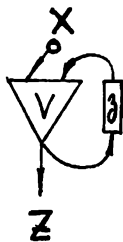


Рис.4.9

Пример 3 (рис. 4.10). Обозначим через А и В выходы конъюнкторов. Выход задержки равен выходу сети в предыдущий момент: $z(t) = Z(t-1)$. Выразим теперь $Z(t)$ через введенные обозначения:

$$A(t) = X(t) \& \bar{Z}(t-1),$$

$$B(t) = X(t) \& Z(t-1),$$

$$Z(t) = A(t) \vee B(t) = (X(t) \vee \bar{Z}(t-1) \vee (\bar{X}(t) \vee Z(t-1) =$$

[известное тождество] $= X(t) \oplus Z(t-1) = X(t) \oplus X(t-1) \oplus Z(t-2) = \dots = X(t) \oplus$

$$X(t-1) \oplus \dots \oplus X(0) = \bigoplus_{\tau=0}^t X(\tau). \text{ Итак, выходное значение меняется после}$$

каждой следующей входной единицы, или, по-другому, $Z(t)$ определяется четностью числа единиц в начальном отрезке входной последовательности.

Используя последнее выражение, можно построить эквивалентную схему (рис. 4.11).

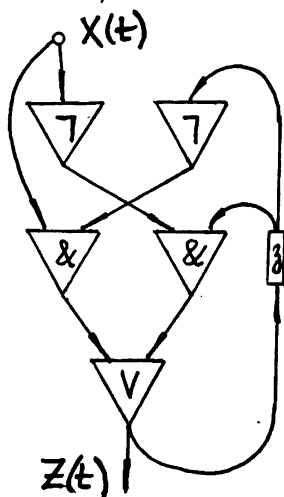


Рис. 4.10

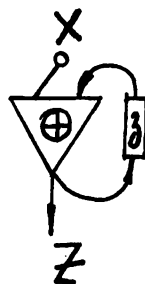


Рис. 4.11

Данный пример показывает, что выходное значение может зависеть от многих (в частности, от всех) предыдущих значений аргумента: так, если изменить значение $X(\tau)$ при одном значении τ , то изменятся все последующие значения $Z(t)$.

Пример 4 (рис. 4.12). Обозначим через Y выход первого сумматора.

Тогда

$$Y(t) = X(t) \oplus Z(t-2),$$

$$Z(t) = Y(t) \oplus Z(t-1).$$

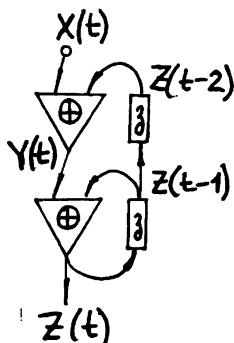


Рис. 4.12

Определим выходную последовательность $Z(t)$ в ответ на некоторые входные последовательности.

а) $X: 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ldots$

$Y: 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ldots$

$Z: 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ldots$

б) $X: 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ldots$

$Y: 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ldots$

$Z: 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ldots$

в) $X: 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ldots$

$Y: 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ldots$

$Z: 1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ldots$

Как показывают рассмотренные примеры, логическая сеть перерабатывает (т.е. отображает) входную последовательность $X(t)$ (или последовательность векторов $(X(t), Y(t))$ и т.п.) в выходную $Z(t)$. Это отображение называется **оператором** $X(t) \rightarrow Z(t)$.

2. На рис. 4.6 построена схема из функциональных элементов для параллельного сложения многозначных двоичных чисел: на $2n$ входов схемы одновременно подаются знаки n -разрядных слагаемых. При увеличении значности слагаемых схему нужно нарастить дополнительными блоками для вычисления старших разрядов суммы. Намного проще выглядит логическая сеть для последовательного сложения. Это схема с 2 входами и 1 выходом; на входы в тактовом режиме подаются (за n тактов) знаки слагаемых X и Y , начиная с младших

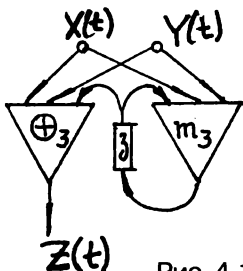


Рис. 4.13

разрядов. На выходе, также начиная с младших разрядов, образуются знаки суммы Z .

Сеть на рис. 4.13 состоит из двух трехвходовых функциональных элементов и одной задержки. Элемент $\oplus_3$ вычисляет очередную цифру суммы $X + Y$. Элемент m_3 определяет знак переноса в старший разряд.

§3. Конечные автоматы

1. **Конечным автоматом** называется система (A, B, Q, φ, ψ) , где $A = \{a_1, a_2, \dots, a_p\}$ - входной алфавит, $B = \{b_1, b_2, \dots, b_r\}$ - выходной алфавит, $Q = \{q_1, q_2, \dots, q_w\}$ - множество внутренних состояний (q_1 называется начальным состоянием); функция выходов $\varphi: Q \times A \rightarrow B$ и функция переходов $\psi: Q \times A \rightarrow Q$ определены на множестве $Q \times A$.

Функционирование автомата происходит в тактовом дискретном режиме для $t = 0, 1, 2, \dots$. На вход автомата поступает входная последовательность $a(0), a(1), a(2), \dots$ ($a(i) \in A$). Пара функций φ, ψ образует **систему канонических уравнений**:

$$b(t) = \varphi(q(t), a(t)),$$

$$q(t+1) = \psi(q(t), a(t)),$$

В. 100

при этом считается, что $q(0) = q_1$.

Первое уравнение задает выходные значения в момент времени t как функцию входного значения в тот же момент времени и текущего внутреннего состояния автомата. Второе уравнение определяет переход в очередное внутреннее состояние, в котором окажется автомат в следующий момент времени. Совокупность $(q(t), a(t))$, полностью определяющая текущее значение выхода и следующее внутреннее состояние автомата, называют полным состоянием автомата.

Основное свойство автомата состоит в следующем: если задать входную последовательность $\{a(t)\}$, то канонические уравнения однозначно определяют выходную последовательность $\{b(t)\}$ и последовательность состояний, $\{q(t)\}$. Действительно, $(q(0), a(0))$ определяют $(q(1), b(0))$, $(q(1), a(1))$ определяют $(q(2), b(1))$ и т.д. Иными словами, автомат осуществляет однозначное отображение входной последовательности в выходную, т.е. реализует оператор $A(t) \rightarrow B(t)$.

Поскольку функции φ и ψ определены на конечных множествах, их можно задать таблицами, которые обычно сводятся в одну таблицу для их декартова произведения $\varphi \times \psi: Q \times A \rightarrow Q \times B$, называемую **таблицей (или матрицей) переходов автомата**; она имеет w строк и p столбцов.

Пример 1. Таблица 4.3 задает функции переходов и выходов для автомата с алфавитами $A = \{a, b, c\}$, $B = \{1, 2\}$, $Q = \{q_1, q_2, q_3, q_4\}$; символы q_i и b_j разделены знаком /.

Таблица 4.3

	a	b	c
q ₁	q ₃ / 1	q ₃ / 2	q ₂ / 1
q ₂	q ₄ / 1	q ₁ / 1	q ₁ / 1
q ₃	q ₂ / 1	q ₃ / 1	q ₃ / 2
q ₄	q ₄ / 1	q ₂ / 1	q ₁ / 2

Автомат можно задавать также **графом** (или **диаграммой переходов** – ориентированным графом (Q, E) , где Q – множество вершин, а дуги соответствуют клеткам таблицы переходов: если на пересечении строки q_i и столбца a_j в таблице содержится пара (q_k / b) , то этой клетке соответствует дуга (q_i, q_k) , которой приписана упорядоченная пара $(a_j; b)$. Граф переходов для таблицы 4.3 изображен на рис. 4.14.

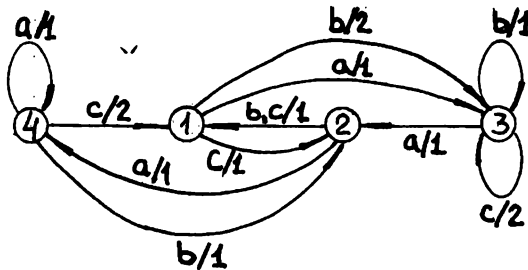


Рис. 4.14

Из каждой вершины графа переходов исходят p дуг, соответствующих столбцам таблицы переходов. Однако для большей наглядности несколько кратных дуг, которым приписаны пары $(a_{i_1}; b)$, $\dots$, $(a_{i_r}; b)$ с одинаковым выходным значением, могут быть склеены в одну дугу, которой приписывается кортеж $(a_{i_1}, \dots, a_{i_r}; b)$. Так, автомат на

рис. 4.14 в ситуациях q_2b и q_2c переходит в одно и то же состояние q_1 при одинаковом выходном значении 1.

Пример 2. Построим автомат, реализующий следующую зависимость входной и выходной последовательностей:

$$Z(t) = \begin{cases} 1, & \text{если число } n(t) = \sum_{\tau=0}^t X(\tau) \text{ делится на 3,} \\ 0 & \text{- в противном случае} \end{cases}$$

Выходное значение зависит от числа единиц в начальном отрезке $[0, t]$ входной последовательности. При этом существенно не общее число единиц $n(t)$, а остаток $rest_3(t)$ от деления $n(t)$ на 3.

Поскольку каждая новая единица во входном наборе сдвигает $rest_3(t)$ на 1 (точнее переводит остаток 0 в 1, остаток 1 в 2, а остаток 2 в 0), то для автомата достаточно иметь 3 состояния, соответствующие различным остаткам от деления на 3: их в этом случае удобно назвать q_0 , q_1 , q_2 , считая начальным состояние q_0 . Диаграмма переходов изображена на рис. 4.15.

При $x = 1$ в каждом из состояний происходит переход в следующее состояние по циклу $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_0$; при $x = 0$ автомат остается в том же состоянии. При этом $Z(t) = 1$ только при переходе в состояние q_0 .

Упражнение. Постройте таблицу переходов этого автомата.

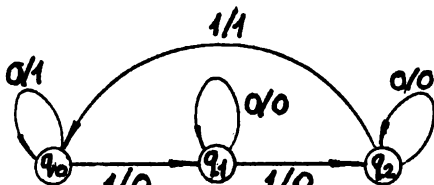


Рис. 4.15

2. Для того чтобы представить канонические уравнения автомата в аналитической форме, т.е. формулами, закодируем символы входного и выходного алфавитов, а также внутренние состояния

двоичными кодами: начальными отрезками натурального ряда в двоичной записи в виде $e_k(n)$. В таком случае входу будет соответствовать вектор $X = (x_1, \dots, x_n)$, выходу - вектор $Z = (z_1, \dots, z_m)$, а состояниям - вектор $Q = (q_1, \dots, q_s)$; причем

$$n = \lceil \log p \rceil, m = \lceil \log r \rceil, s = \lceil \log w \rceil.$$

При таком кодировании канонические уравнения автомата принимают вид:

$$Z(t) = \tilde{\Phi}(Q(t), \tilde{X}(t))$$

$$Q(t+1) = \tilde{\Psi}(Q(t), \tilde{X}(t)),$$

где x_i, z_j, q_k - булевы переменные, а $\tilde{\Phi}$ и $\tilde{\Psi}$ - булевы вектор-функции:

$$\tilde{\Phi} = (\varphi_1, \dots, \varphi_m), \quad \tilde{\Psi} = (\psi_1, \dots, \psi_s).$$

Пример. Для автомата, заданного таблицей 4.3 (и рис. 4.14), кодирование таково: $a \leftrightarrow 00, b \leftrightarrow 01, c \leftrightarrow 10; q_1 \leftrightarrow 00, q_2 \leftrightarrow 01, q_3 \leftrightarrow 10, q_4 \leftrightarrow 11; 1 \leftrightarrow 0, 2 \leftrightarrow 1$. Таблица переходов выглядит так:

	00	01	10
00	10 / 0	10 / 1	01 / 0
01	11 / 0	00 / 0	00 / 0
10	01 / 0	10 / 0	10 / 1
11	11 / 0	01 / 0	00 / 1

Аналогично примеру §1 п. 3 составим таблицу функций $Z(t), q(t+1)$.

Поскольку входной алфавит имеет 3 значения, то при двоичном кодировании вектор 11 не используется. Для выражения функций формулами нужно произвольно доопределить их и для наборов $(q_1, q_2, 1, 1)$. В таблице 4.4 доопределение выбрано так, чтобы формула оказалась возможно проще.

Функцию $Z(t)$ можно записать, используя разложение по первым двум переменным:

$$Z = \bar{q}_1 \cdot \bar{q}_2 \cdot x_2 \vee q_1 \cdot \bar{q}_2 \cdot x_1 \vee q_1 \cdot q_2 \cdot x_1 = \bar{q}_1 \cdot \bar{q}_2 \cdot x_2 \vee q_1 \cdot x_1.$$

Упражнение. Получите аналогичные формулы для $q_1(t+1)$ и $q_2(t+1)$.

3. Как пример построения автомата рассмотрим схему, крайне упрощенно описывающую процедуру, имитирующую выработку классического условного рефлекса (таблица 4.5, рис. 4.16). Система (модель, представляющая животное) имеет два входа: X - безусловный раздражитель - пища, Y - условный раздражитель - звонок (1 - наличие, 0 - отсутствие). 4 столбца таблицы переходов соответствуют

Таблица 4.4

q(t)	X(t)	q(t+1)	Z(t)	Доопреде- ление q	Доопреде- ление Z	q после доопре- деления	Z после доопре- деления
q1, q2	x1, x2						
0 0	0 0	1 0	0			1 0	0
0 0	0 1	1 0	1			1 0	1
0 0	1 0	0 1	0			0 1	0
0 0	1 1	- -	-	0 1	1	0 1	1
- -	- -	- -	- -	- -	- -	- -	- -
0 1	0 0	1 1	0			1 1	0
0 1	0 1	0 0	0			0 0	0
0 1	1 0	0 0	0			0 0	0
0 1	1 1	- -	-	1 1	0	1 1	0
- -	- -	- -	- -	- -	- -	- -	- -
1 0	0 0	0 1	0			0 1	0
1 0	0 1	1 0	0			1 0	0
1 0	1 0	1 0	1			1 0	1
1 0	1 1	- -	-	0 1	1	0 1	1
- -	- -	- -	- -	- -	- -	- -	- -
1 1	0 0	1 1	0			1 1	0
1 1	0 1	0 1	0			0 1	0
1 1	1 0	0 0	1			0 0	1
1 1	1 1	- -	-	1 0	1	1 0	1

всевозможным сочетаниям этих факторов. 5 состояний автомата $q_1 \div q_5$ отражают 4 этапа формирования рефлекса:

q_1 - начальное состояние;

q_2, q_3 - этапы постепенного обучения;

q_4 - этап выработанного рефлекса;

q_5 - этап повторной реакции и угасания степени обучения.

Выходное значение Z - реакция на раздражители ($Z = 1$ - слюноотделение, $Z = 0$ - его отсутствие).

При наличии обоих факторов ($X = Y = 1$) происходит процесс обучения (постепенный переход в следующее состояние) $q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4$.

Если в этом процессе не происходит подкрепления, то система откатывается в предыдущее состояние (частичное забывание). При этом, если $X = 1$, то $Z = 1$ при любом значении Y (столбцы 10 и 11). Если $X = 0$, то $Z = 0$ в состояниях q_1, q_2, q_3 вплоть до состояния q_4 (для простоты мы

предусматриваем только 2. этапа обучения - можно было бы удлинить эту серию). В состоянии q_4 при $Y = 1$ (звонок) $Z = 1$ даже при $X = 0$ (отделяется слюна даже при отсутствии пищи). В состоянии q_5 выходное значение $Z(01)$ равно 1 как бы по инерции, но происходит забывание разной степени в зависимости от значений входных переменных.

Таблица 4.5

	00	01	10	11
q1	q1 / 0	q1 / 0	q1 / 1	q2 / 1
q2	q1 / 0	q1 / 0	q1 / 1	q3 / 1
q3	q2 / 0	q2 / 0	q2 / 1	q4 / 1
q4	q3 / 0	q4 / 1	q3 / 1	q4 / 1
q5	q3 / 0	q2 / 1	q3 / 1	q4 / 1

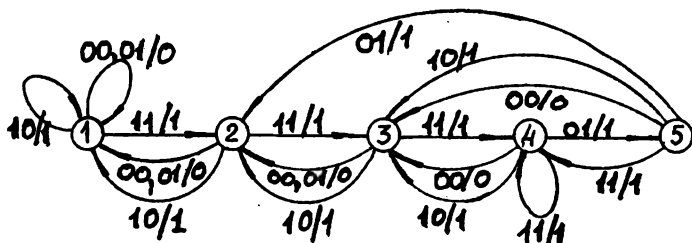


Рис. 4.16

4. Эквивалентность логической сети и автомата.

Назовем автоматы $(A_1, B_1, Q_1, \phi_1, \psi_1)$ и $(A_2, B_2, Q_2, \phi_2, \psi_2)$ эквивалентными, если существует такое взаимно однозначное соответствие Γ между их элементами: любым входным наборам $X_1(t)$ и $X_2(t)$, для которых $X_2(t) = \Gamma(X_1(t))$, соответствуют выходные наборы $Z_1(t)$ и $Z_2(t)$, для которых $Z_2(t) = \Gamma(Z_1(t))$.

Можно утверждать, что логическая сеть является конечным автоматом. Вход сети естественно рассматривать как вход, а выход сети - как выход автомата. Нужно определить, что называть состоянием сети, чтобы выполнялось определение автомата.

Пусть логическая сеть S содержит n входов $X_1, \dots, X_n$, m выходов $Z_1, \dots, Z_m$.

..., Z_m и v задержек $z_1, \dots, z_v$. Будем рассматривать вход сети $\tilde{X} = (X_1, \dots, X_n)$ как вход автомата, выход сети $\tilde{Z} = (Z_1, \dots, Z_m)$ - как выход автомата. За состояние автомата можно принять вектор $(z_1, \dots, z_v)$ значений выходов задержек.

Действительно, если в некоторый момент t известны значения на выходах всех задержек сети $\{z_i(t)\}$ и входное значение $\tilde{X}(t)$, то однозначно определены, во-первых, значения на выходах всех элементов сети и тем самым выходной вектор $\tilde{Z}(t)$ и значения на входах всех задержек в этот момент времени, а значит, во-вторых, значения на выходах задержек в следующий момент времени $z_i(t+1)$, т.е. то, что мы обозначили как состояние автомата.

Мы показали, что для любой логической сети существует эквивалентный ей автомат.

Для логической сети можно построить таблицу переходов, граф переходов, канонические уравнения.

Верно и обратное утверждение.

Теорема. Для конечного автомата можно построить эквивалентную ему логическую сеть.

В п. 2 предыдущего параграфа показано, как перевести канонические уравнения автомата в двоичную форму (*), где зависимость выходных значений и переходов выражена булевыми функциями входных значений и состояния автомата. В этом случае логическая сеть строится так, как показано на рис. 4.17. Схема из функциональных элементов S реализует совокупность $(m + s)$ булевых функций от $(n + s)$ булевых переменных:

$$\varphi_i(q_1, \dots, q_s, X_1, \dots, X_n), i = 1, \dots, m$$

$$\psi_j(q_1, \dots, q_s, X_1, \dots, X_n), j = 1, \dots, s.$$

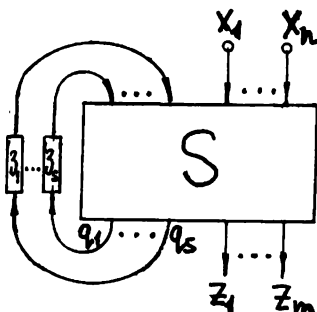


Рис. 4.17

В момент времени $t = 0$ выходы всех задержек, равные 0, поступают одновременно с значениями $\{X_k(0)\}$ на входы схемы $S(k = 1, \dots, n)$. Набор $\{z_j(0)\}$ соответствует начальному состоянию q_1 автомата. Схема вычисляет выходные значения $\{Z_j(0)\}$, а также значения $\{q_j(1)\}$, которые в текущий момент времени поступают на входы задержек, а в следующий момент времени оказываются на выходах задержек и соответствуют состоянию автомата в этот момент ($t = 1$). Таким же образом функционирует логическая сеть S и в моменты времени $t = 1, 2, \dots$

Понятия логической сети и автомата демонстрируют два различных подхода при изучении функционирования системы. Логическая сеть изображает внутреннее устройство системы, ее функциональный состав и структуру; автомат представляет подход, обозначаемый в кибернетике как “черный ящик”, когда о системе судят по ее реакциям на внешние воздействия.

Можно сформулировать две основные задачи теории автоматов: анализ и синтез. При анализе исследуют, как функционирует заданная система, как входные последовательности преобразуются в выходные, проводя их по цепям функциональных элементов для логической сети или по диаграмме для автомата. Задача синтеза - построение схемы или автомата, обеспечивающих нужное функционирование, подобно задачам в примере 2 п. 2 или в п. 3.

§4. Периодические последовательности в автомате

1. Последовательность $\alpha(1), \alpha(2), \dots, \alpha(n)$ называется периодической, если

$$\exists t_0, T: \forall t > t_0: \alpha(t) = \alpha(t+T). \quad (*)$$

Вместе с числом T этому условию обязательно удовлетворяют числа $2T, 3T, \dots$. Периодом последовательности называется минимальное положительное T , а предпериодом - минимальное t_0 , удовлетворяющие условию (*). Если $t_0 = 0$, последовательность называется чисто периодической. Величина $t_0 + T$ называется полной длиной периода.

Для $d = 1, 2, \dots, T$ и любого $k = 0, 1, 2, \dots$ выполнено равенство:

$$\alpha(t_0+d) = \alpha(t_0+d+kT), \quad (**)$$

тем самым d определяет фазу внутри периода.

Теорема. Пусть A - автомат, имеющий s внутренних состояний, на вход которого подается периодическая последовательность с параметрами $t_0(X) = \mu$, $T(X) = \omega$. Тогда выходная последовательность - также периодическая с периодом $T(Z) \leq s \cdot \omega$, а полная длина периода $t_0(Z) + T(Z) \leq \mu + s \cdot \omega$.

Доказательство. По комбинаторному правилу произведения число различных полных состояний автомата, определяемое сочетанием не более чем ω различных входных значений и s различных внутренних состояний, не превосходит $s \cdot \omega$. Рассмотрим функционирование автомата на протяжении $(s \cdot \omega + 1)$ тактов, начиная с $t = \mu + 1$ (с этого момента входная последовательность - чисто периодическая). Тогда, по принципу Дирихле, среди $(s + 1)$ моментов времени $\mu + 1$, $\mu + \omega + 1$, $\mu + 2\omega + 1, \dots$, $\mu + s\omega + 1$ существуют два таких $t_1 = \mu + k_1\omega + 1$ и $t_2 = \mu + k_2\omega + 1$, что $q(t_1) = q(t_2)$. Поскольку t_1 и t_2 - в одной фазе, то $X(t_1) = X(t_2)$ и, следовательно, $Z(t_1) = Z(t_2)$, $q(t_1 + 1) = q(t_2 + 1)$, т.к. пара $(X(t), q(t))$ определяет $Z(t)$ и $q(t+1)$.

Моменты времени $t_1 + 1$ и $t_2 + 1$ - также в одной фазе, поэтому $Z(t_1 + 1) = Z(t_2 + 1)$, $q(t_1 + 2) = q(t_2 + 2)$ и т.д. Отсюда $T = t_2 - t_1$ есть период или кратное периода, что и доказывает теорему, поскольку $T \leq (k_2 - k_1) \cdot \omega$.

Упражнение. Рассмотрите с этой точки зрения пример 4 в п. 1 § 2.

2. Некоторые следствия.

Можно рассмотреть автоматы без входов, называемые автономными, на которые как бы не воздействует внешняя среда (пример - часы). Следующее состояние автономного автомата полностью определяется предыдущим.

Функционирование такого автомата такое же, как у автомата с одним входом, на который подается константа. Ясно, что автономный автомат также выдает на выходе периодическую последовательность $Z(t) = \Phi(q(t))$.

Граф переходов отличается тем, что из каждой его вершины исходит ровно одна дуга, и путь, начинающийся в вершине q , в некоторый момент попадает второй раз в уже пройденную вершину и с этого момента зацикливается. Поэтому длина этого цикла кратна периоду выходной последовательности, откуда следует, что этот период не превосходит числа внутренних состояний автомата (рис. 4.18).

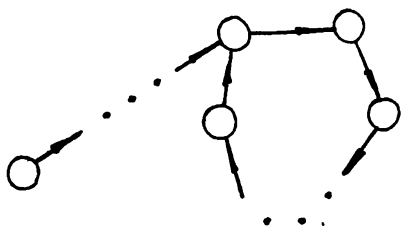


Рис. 4.18

Можно сказать, что разнообразие реакций автомата ограничено числом его полных состояний, и в этом смысле автомат обладает конечной памятью. В связи с этим рассмотрим следующий пример.

В §2 описана логическая сеть (и тем самым автомат), реализующая последовательное сложение

многозначных двоичных чисел. Можно показать, что аналогичный автомат для умножения не существует, хотя каждый знак произведения зависит только от предыдущих знаков сомножителей (умножение, как и сложение, производится, начиная с младших разрядов).

Уточним постановку задачи. Автомат M для умножения должен функционировать следующим образом. На два входа X и Y синхронно подаются два многозначных числа, начиная с младших разрядов. После ввода старшего разряда любого из чисел на соответствующий вход подаются нули (произведение n -значных чисел есть $2n$ -значное число, и его расчет заканчивается через $2n$ тактов).

Доказательство невозможности построения автомата M , способного перемножать n -значные числа для любого n , - от противного. Если автомат M существует и имеет s состояний, то он может, в частности, перемножить два одинаковых числа, равных 2^n (результат - 2^{2n}), т.е. перевести пару $X = Y = \underbrace{0 \dots 0}_n 100 \dots$ в $Z = \underbrace{0 \dots 0}_{2n} 100 \dots$. Входная

последовательность имеет предпериод $t_0(X) = n + 1$ и период $\omega(X) = 1$; для выходной последовательности $t_0(Z) = 2n + 1$, $\omega(Z) = 1$. По теореме о периодической последовательности в автомате, полная длина периода выходной последовательности $t_0(Z) + T(Z) \leq n + 1 + s \cdot 1$. В то же время она равна $t_0(Z) + \omega(Z) = 2n + 2$. Но если $n \geq s$, то $2n + 2 > n + 1 + s$. Противоречие, доказывающее несуществование автомата A .

Полученный результат можно интерпретировать таким образом: при умножении чисел приходится запоминать большое число промежуточных величин, от которых зависит результат, и, как показывает приведенное доказательство, их невозможно свести к ограниченному числу, как это удастся в случае сложения чисел, где достаточно запоминать один знак переноса в следующий разряд.

Аналогично, невозможно реализовать конечным автоматом оператор

$$Z(t) = \begin{cases} 1, & \text{если } \sum_{\tau=0}^t X(\tau) > t/2, \\ 0 & - \text{в противном случае,} \end{cases}$$

т.е. $Z(t) = 1$, если в начальном отрезке $[0, t]$ входной последовательности число единиц больше числа нулей. Для определения $Z(t)$ необходимо в каждый момент времени помнить, на сколько число предшествующих единиц отличается от числа нулей, т.е. уметь запоминать как угодно большие числа, а это невозможно при ограниченной памяти. Доказательство может быть проведено таким же образом, как в предыдущем примере, опираясь на теорему о периодической последовательности в автомате.

ГЛАВА V. АЛГОРИТМЫ И ВЫЧИСЛИМОСТЬ

§1. Требования к алгоритмическим процедурам

1. Зададимся вопросом: что значит задать функцию? Один из возможных конструктивных ответов таков: функция считается заданной, если задана вычислительная процедура, которая по любому заданному (допустимому) значению аргумента позволяет получить соответствующее значение функции. Функцию, определяемую таким образом, можно назвать вычислимой. Процедура должна однозначно приводить к результату. Уточнение различных связанных с этим понятий привело к разработке теории алгоритмов.

Под **алгоритмом** понимается точное предписание, которое задает вычислительный в широком смысле процесс, называемый в этом случае алгоритмическим. Процесс начинается с произвольного исходного данного из некоторой совокупности допустимых для данного алгоритма исходных данных и направлен на получение результата, полностью определяемого этим исходным данным.

Школьные методы умножения “столбиком” и деления “углом” многозначных десятичных чисел, метод исключения неизвестных при решении системы линейных алгебраических уравнений, правило дифференцирования сложной функции - примеры вычислительных алгоритмов.

Само слово “алгоритм” происходит от *algoritmi* - имени (в латинском написании) арабского математика IX в. аль-Хорезми. В средневековой Европе алгоритмом называлась десятичная позиционная система счисления и искусство счета в ней, поскольку именно благодаря

латинскому переводу (XII в.) трактата аль-Хорезми Европа познакомилась с позиционной системой счисления.

Вообще говоря, не предполагается, что результат будет обязательно получен: процесс применения алгоритма к конкретному возможному исходному данному (т.е. алгоритмический процесс, развертывающийся начиная с этого данного) может также оборваться безрезультатно (в этом случае говорят о **безрезультатной остановке**) или не закончиться вовсе. В случае, если процесс заканчивается (соответственно не заканчивается) получением результата, говорят, что алгоритм **применим** (соответственно **неприменим**) к рассматриваемому исходному данному.

Вычислительный процесс не следует понимать в узком смысле только цифровых вычислений. Целесообразно рассматривать алгоритмы, оперирующие с произвольными символами и их комбинациями. Простейшим случаем такой комбинации является кортеж символов, образующий слово в алфавите, однако можно рассматривать и "нелинейные" комбинации - такие, как алгебраические матрицы, фразы того или иного языка с расставленными стрелками синтаксического управления и, вообще, размеченные тем или иным способом графы. Именно поэтому понятие алгоритма является одним из основных понятий в современной математике и кибернетике. С точки зрения современной практики алгоритм - это программа, а критерием алгоритмичности процесса является возможность его запрограммировать.

Необходимо уметь сравнивать различные алгоритмы решения одних и тех же задач, причем не только по качеству решения, но и по характеристикам самих алгоритмов (числу действий, расходу памяти и др.). Это, однако, невозможно без введения точных, а не просто интуитивных понятий, т.е. сами алгоритмы должны стать такими же предметами точного исследования, как и те объекты, для работы с которыми они предназначены.

2. Основные требования к алгоритмам.

А. Как сказано выше, алгоритм обрабатывает данные. Требуется уточнить понятие о данных, которые должны быть четко определены.

В теории алгоритмов фиксируют конечные наборы исходных объектов в виде конечного алфавита исходных символов (цифр, букв и т.д.) и конечного набора средств построения других объектов из элементарных; типичным средством являются индуктивные определения. В процессе работы алгоритма образуются промежуточные данные, которые также подвергаются алгоритмической обработке.

Б. Алгоритм состоит из отдельных *элементарных шагов, или действий*,

причем множество различных шагов, из которых составлен алгоритм, конечно. Последовательность шагов алгоритма детерминирована, т.е. после каждого шага указывается, какой шаг делать дальше, либо дается команда *остановки*, после которой работа алгоритма считается законченной.

В. От алгоритма требуется *результативность*, т.е. остановка после конечного числа шагов (зависящего от исходных данных) с указанием того, что считать результатом. В частности, чтобы показать, что некоторый алгоритм решает определенную задачу, необходимо убедиться, что алгоритм останавливается после конечного числа шагов (как говорят, *сходится*) для всех интересующих нас исходных данных этой задачи.

Г. Рассмотрение алгоритма как последовательности элементарных шагов сопряжено с ограниченностью изменений при каждом шаге, иначе говоря, *локальностью* преобразований. Это значит, что преобразование исходного или промежуточного объекта складывается из изменений в отдельных его частях: для многозначных чисел - в соседних разрядах, для слов в алфавите - в некотором подслове ограниченной длины, для графа - в некоторой окрестности отдельной вершины (подграфе ограниченных размеров). В этих случаях можно ввести правила локальных преобразований небольшого (по сравнению с возможными размерами объекта) количества типов.

В гл. I - III рассматривались некоторые процедуры. Формулы, в том числе комбинаторные, можно считать алгоритмами в интуитивном смысле: некоторые из них содержат конечное число (арифметических) действий, не зависящее от значения аргумента, другие, например, содержащие обозначение $n!$, - подразумевают вполне определенный процесс вычислений. Стратегия (любая) в детерминированной игре с открытой информацией есть алгоритм. Построение остова графа последовательностью удалений ребра из цикла, как оно описано в гл. III, не удовлетворяет требованию детерминированности, т.к., во-первых, не объяснено, как выбирается очередной элементарный цикл, во-вторых, выбор такого цикла может быть не однозначным и, в-третьих, неоднозначен выбор удаляемого ребра. Можно, однако, каким-либо образом упорядочить циклы (например, упорядочив ребра графа $e_1, e_2, \dots, e_p$, записывать цикл как слово в алфавите $\{e_i\}$ и перебирать циклы в алфавитном порядке). То же относится к построению базиса циклов - путем упорядочивания множества хорд относительно выбранного остова, а также к процессу отыскания кратчайших путей (цепей). Построение кода Фано в некоторых случаях не детерминировано из-за возможности неоднозначного деления списка букв, однако и здесь нетрудно уточнить процедуру выбора и сделать метод Фано алгоритмическим.

3. *Пример.* Пусть возможными исходными данными служат любые слова в алфавите $\{a, b\}$. Будем считать переход от слова α к слову β допустимым в двух случаях (γ, δ означают произвольные слова):

(1) $\alpha = a\gamma$, $\beta = \gamma b$, т.е. начальный символ a стирается, а справа к слову приписывается символ b ;

(2) $\alpha = ba\gamma$, $\beta = \gamma aba$, т.е. в начале слова стирается подслово ba , справа приписывается подслово aba .

Алгоритм (обозначим его A) состоит в следующем: “взяв какое-либо слово в качестве исходного, делай допустимые переходы до тех пор, пока не получится слово вида aad , - это признак окончания процедуры, а слово δ является ее результатом”.

Рассмотрим 3 примера применения алгоритма A к различным исходным данным. Можно заметить, что к любому слову может быть применимо только одно из правил перехода: (1) - если слово начинается символом a ; (2) - если начало слова - символ b .

1) Исходное слово $babaa$. После 1-го перехода получаем $baaaba$, после 2-го - $aabaaba$. В силу предписания мы должны остановиться: результат есть $baaba$.

2) Исходное слово $baaba$. Получим последовательно

по 2-му правилу: $abaaba$

по 1-му правилу: $baabab$

по 2-му правилу: $abababa$

по 1-му правилу: $bababab$

по 2-му правилу: $babababa...$

Можно показать, что процесс никогда не кончится, т.е. никогда не получится слово, начинающееся с aa , и к каждому из получающихся слов можно будет применить правило (1) или (2) в зависимости от начального символа слова.

3) Исходное слово $abaab$. Получим

по 1-му правилу: $baabb$

по 2-му правилу: $abbaba$

по 1-му правилу: $bbabab$.

Далее ни 1-е, ни 2-е правила неприменимы, в то же время нет сигнала остановки (начало aa). Произошла безрезультатная остановка.

Таким образом, алгоритм A применим к слову $babaa$ и неприменим к словам $baaba$ и $abaab$.

4. Алгоритмы встречаются в науке и ее приложениях постоянно: умение решать какую-то задачу “в общем виде” означает, по существу, владение некоторым алгоритмом. Например, говоря об умении человека складывать числа, имеют в виду не то, что он для любых чисел сумеет

рано или поздно найти их сумму, а то, что он владеет некоторым единообразным приемом, применимым к любым двум конкретным записям чисел. Примером такого алгоритма служит известное правило сложения многозначных чисел “столбиком”.

Понятие задачи “в общем виде” уточняется при помощи понятия **массовая алгоритмическая проблема**, в которой требуется найти единый метод (алгоритм) для решения бесконечной серии однотипных единичных задач. Когда такого алгоритма не существует, говорят, что данная массовая проблема неразрешима.

Уточнение понятия алгоритма как однозначной результативной процедуры предпринималось математиками в XX в. различными путями, отталкиваясь от многих эвристических концепций алгоритмического процесса: как вычисление значений числовых функций, как локальные преобразования линейно упорядоченных систем и других объектов, как формализация пошаговой инструкции выполнения последовательности простейших операций.

Мы рассмотрим два подхода к точному понятию алгоритма: рекурсивные функции и машины Тьюринга.

При любом построении точного понятия алгоритма принимается такая схема: выбирается конечный набор исходных объектов, которые объявляются элементарными, и конечный набор способов построения из них новых объектов. В принятых терминах показывают, как разнообразные объекты, которые интуитивно считаются вычислимыми, могут быть получены предлагаемыми средствами. При этом, однако, возможно сомнение в универсальности каждого такого описания: всякую ли процедуру, интуитивно полагаемую алгоритмом, можно реализовать в предлагаемой точной модели? Важно понимать, что ответ на это вопрос не может быть дан в форме математического доказательства, т.к. сопоставляются точно определенная структура и интуитивное представление. Однако доводами в пользу такой адекватности служат три обстоятельства.

(1) Доказана сводимость одних моделей к другим, т.е. показано, что всякий алгоритм, описанный средствами одной модели, может быть описан средствами другой. Поскольку при этом оперируют с точными математическими объектами, то такое сведение осуществляется именно доказательством указанной эквивалентности. Это дает возможность для решения различных задач теории алгоритмов использовать то один, то другой язык, удобный в конкретном случае.

(2) Благодаря взаимной сводимости моделей в теории алгоритмов удалось выработать инвариантную по отношению к отдельным моделям систему понятий, позволяющую говорить о свойствах алгоритмов

независимо от того, какая формализация алгоритма выбрана. Эта система понятий основана на точном понятии **вычислимой функции** как функции, для вычисления которой существует алгоритм.

(3) За более чем полвека, прошедшие со времени разработки основных моделей алгоритма, не возникло сомнений, что всевозможные интуитивно алгоритмические процедуры укладываются в рамки предложенных моделей, и не обнаружилось противоречащих примеров.

Последнее соображение не является, конечно, доказательством. Поэтому каждое построение алгоритмической модели состоит в описании средств и демонстрации возможностей модели, сопоставлении ее с другими моделями (доказательство эквивалентности) и завершается провозглашением тезиса об универсальности данной модели.

§2. Рекурсивные функции

1. Понятие примитивно рекурсивной функции.

Будем рассматривать функции $y = f(x_1, \dots, x_n)$, $x_1, \dots, x_n, y \in \mathbb{N}^0$, где $\mathbb{N}^0 = \{0, 1, \dots, n, \dots\}$ - множество натуральных (целых неотрицательных) чисел; таким образом, и аргументы, и функция принимают натуральные значения.

Следующие функции называются **исходными**:

1) $Z(x) = 0$ - нуль-функция ("zero"), т.е. константа 0;

2) $N(x) = "x + 1"$ - функция следования ("next"), обозначение: x' ;

3) $I_k(x_1, \dots, x_n) = x_k$ - селекторные функции, где $k = 1, 2, \dots, n$.

Функция $N(x)$ выражает число, следующее в натуральном ряду за x . Она представляет собой функцию одной переменной, поэтому запись ее с помощью знака "+" служит только для пояснения (что такое сумма - еще не определено, см. ниже). Каждая из функций $I_k(x)$ равна одному из своих аргументов: индекс k выделяет k -ю переменную.

Из исходных функций мы будем образовывать многие другие функции с помощью операций, которые можно называть функциональными операторами.

Говорят, что функция h получается применением **оператора суперпозиции S** к функциям $f, g_1, \dots, g_n$ (обозначение: $h = S(f, g_1, \dots, g_n)$), если

$$h(x_1, \dots, x_m) = f(g_1(x_1, \dots, x_m), \dots, g_n(x_1, \dots, x_m)).$$

С понятием суперпозиции (или подстановки) мы встречались ранее неоднократно.

Примеры. Целочисленные константы вводятся суперпозициями исходных функций $Z(x)$ и $N(x)$: $N(Z(x)) = N(0) = 1$; $N(N(Z(x))) = N(1) = 2$ и т.д. $Z(f(x)) = 0$ для любой функции $f(x)$. $I_1(3, 2)$ следует рассматривать как сокращенное обозначение выражения $I_1(N(N(N(Z(x))))$, $N(N(Z(x))))$, равного 3; $I_2(3, 2) = 2$. Из функции $f(x, y)$ можно получить функцию $g(x, y) = f(y, x)$ (перестановка переменных) следующей суперпозицией: $g(x, y) = f(I_2(f(x, y)), I_1(f(x, y)))$.

Говорят, что функция f от $(n+1)$ переменных получается применением **оператора примитивной рекурсии R** к функциям g от n переменных и h от $(n+2)$ переменных (обозначение: $f = R(g, h)$), если

- 1) $f(x_1, \dots, x_n, 0) = g(x_1, \dots, x_n)$,
- 2) $f(x_1, \dots, x_n, k+1) = h(x_1, \dots, x_n, k, f(x_1, \dots, x_n, k))$, $k \in N^0$.

Рекурсия в этом определении ведется по последней переменной функции f и выражает встречавшийся ранее индуктивный способ определения: зная значение функции при $k = 0$, мы можем вычислить значение функции для $k = 1$, затем для $k = 2$ и т.д.

Примеры. 1) Для $n = 1$.

$$\begin{cases} f(x, 0) = x \\ f(x, y+1) = f(x, y) + x + 3y \end{cases}$$

$$f(x, 1) = x + x + 0 = 2x,$$

$$f(x, 2) = 2x + x + 3 = 3x + 3,$$

$$f(x, 3) = (3x + 3) + x + 3 \cdot 2 = 4x + 9,$$

$$f(x, 4) = (4x + 9) + x + 3 \cdot 3 = 5x + 18,$$

$$f(x, 5) = (5x + 18) + x + 3 \cdot 4 = 6x + 30.$$

.....

Упражнение. Докажите методом математической индукции, что

$$f(x, n) = (n + 1)x + 3 \cdot C_n^2$$

2) Для $n = 2$.

$$\begin{cases} f(x, y, 0) = 1 \\ f(x, y, z+1) = (x + y) \cdot f(x, y, z) \end{cases}$$

$$f(x, y, 1) = (x + y) \cdot 1,$$

$$f(x, y, 2) = (x + y) \cdot (x + y) = (x + y)^2,$$

$$f(x, y, 3) = (x + y) \cdot (x + y)^2 = (x + y)^3$$

.....

Нетрудно заметить, что $f(x, y, n) = (x + y)^n$.

3) Для $n = 2$.

$$\begin{cases} f(x, y, 0) = 1 \\ f(x, y, z + 1) = (x + z) \cdot f(x, y, z) \end{cases}$$

$$f(x, y, 1) = (x + 0) \cdot 1 = x,$$

$$f(x, y, 2) = (x + 1) \cdot x,$$

$$f(x, y, 3) = (x + 2) \cdot (x + 1) \cdot x.$$

.....

Также легко доказать, что $f(x, y, n) = \prod_{k=0}^{n-1} (x + k)$.

4) Для $n = 0$.

$$\begin{cases} f(0) = 0 \\ f(x + 1) = f(x) + 2x + 1 \end{cases}$$

$$f(1) = 0 + 2 \cdot 0 + 1 = 1,$$

$$f(2) = 1 + 2 \cdot 1 + 1 = 4,$$

$$f(3) = 4 + 2 \cdot 2 + 1 = 9,$$

$$f(4) = 9 + 2 \cdot 3 + 1 = 16.$$

.....

Методом математической индукции легко показать, что $f(n) = n^2$, поскольку $f(n + 1) = n^2 + 2n + 1$.

5) Для $n = 0$.

$$\begin{cases} f(0) = 0 \\ f(x + 1) = x \cdot f(x) + 2 \end{cases}$$

$$f(1) = 0 \cdot 0 + 2 = 2,$$

$$f(2) = 1 \cdot 2 + 2 = 4,$$

$$f(3) = 2 \cdot 4 + 2 = 10,$$

$$f(4) = 3 \cdot 10 + 2 = 32,$$

$$f(5) = 4 \cdot 32 + 2 = 130,$$

$$f(6) = 5 \cdot 130 + 2 = 652,$$

$$f(7) = 6 \cdot 652 + 2 = 3914.$$

Чтобы получить явное выражение для $f(n)$, запишем рекуррентную

формулу в виде $f(x + 1) = 2 + f(x) \cdot x$. Тогда можно переписать предыдущие равенства:

$$f(1) = 2 + 0 \cdot 0,$$

$$f(2) = 2 + 2 \cdot 1,$$

$$f(3) = 2 + (2 + 2 \cdot 1) \cdot 2 = 2 \cdot \{1 + (1 + 1) \cdot 2\},$$

$$f(4) = 2 + (2 + (2 + 2 \cdot 1) \cdot 2) \cdot 3 = 2 \cdot \{1 + (1 + (1 + 1) \cdot 2) \cdot 3\},$$

$$f(5) = 2 + (2 + (2 + (2 + 2 \cdot 1) \cdot 2) \cdot 3) \cdot 4 = 2 \cdot \{1 + (1 + (1 + (1 + 1) \cdot 2) \cdot 3) \cdot 4\},$$

$$f(6) = 2 + (2 + (2 + (2 + (2 + 2 \cdot 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5 = 2 \cdot \{1 + (1 + (1 + (1 + (1 + 1) \cdot 2) \cdot 3) \cdot 4) \cdot 5\},$$

$$f(n) = 2 + (2 + (2 + (2 + \dots + 2 \cdot 1) \cdot 2) \cdot 3) \cdot \dots \cdot (n-1) = 2 \cdot \{1 + (1 + (1 + (1 + \dots + 1) \cdot 2) \cdot 3) \cdot \dots \cdot (n-1)\}.$$

Разделим последнее выражение на $(n - 1)!$:

$$\frac{f(n)}{(n-1)!} = 2 \cdot \left(\frac{1}{(n-1)!} + \frac{1}{(n-2)!} + \dots + \frac{1}{1!} + \frac{1}{0!} \right) \approx 2 \cdot e,$$

откуда $f(n) \approx 2 \cdot e \cdot (n - 1)!$, и можно показать (так как $f(n)$ – целое число), что $f(n) = [2 \cdot e \cdot (n - 1)!]$.

2. Теперь введем одно из основных понятий этого раздела.

Функция называется **примитивно рекурсивной** (сокращенно **п/р**), если она может быть получена из исходных применением конечного числа операторов суперпозиции и примитивной рекурсии.

Отметим, что п/р функции определены при всех $x \in \mathbb{N}^0$.

Рассмотрим некоторые примеры примитивно рекурсивных функций.

Функции, выражающие основные арифметические действия сложения и умножения, являются п/р.

А) Сумма $S(x, y) = x + y$ может быть определена так:

$$1) S(x, 0) = x.$$

Точнее, в обозначениях определения 1, $S(x, 0) = I_1(x)$,

2) $S(x, y + 1) = N(S(x, y)) = (S(x, y))'$, т.е. сумма $x + (y + 1)$ равна числу, следующему за $(x + y)$: $(x + y) + 1$.

В) Произведение $P(x, y) = x \cdot y$ может быть определено так:

$$1) P(x, 0) = Z(x), \text{ т.е. } x \cdot 0 = 0$$

$$2) P(x, y + 1) = S(P(x, y), x), \text{ т.е. } x \cdot (y + 1) = x \cdot y + x$$

Поскольку операции вычитания, деления и возведения в степень не всегда выполнимы в области натуральных чисел, вводятся их всюду определенные аналоги.

Усеченной разностью называется функция $x \dot{-} y = \begin{cases} x - y, & \text{если } x \geq y, \\ 0, & \text{если } x < y. \end{cases}$

Функция $x \dot{-} y$ принимает натуральные значения при всех x, y . Обозначим через $\delta(x)$ функцию одной переменной $\delta(x) = x \dot{-} 1$; Положим $x^y = 1$, если $x = y = 0$.

Можно показать, что функции $(x \dot{-} y)$, $\delta(x)$, x^y , $|x - y|$ - примитивно рекурсивны. Также являются п/р функции $[x / y]$ - целая часть частного и $\text{rest}(x, y)$ - остаток от деления x на y , если положить $[x / 0] = x$, $\text{rest}(x, 0) = x$ для всех x .

Примеры. $6 \dot{-} 4 = 2$; $4 \dot{-} 6 = 0$; $[3/7] = 0$; $[7/3] = 2$; $[7/2] = 3$; $\delta(0) = \delta(1) = 0$; $\delta(5) = 4$.

3. Примитивно рекурсивные предикаты. Условный оператор.

Предикат $P(x_1, \dots, x_n)$ называется примитивно рекурсивным, если является п/р его характеристическая функция

$$\chi_p(x_1, \dots, x_n) = \begin{cases} 0, & \text{если } P(x_1, \dots, x_n) \text{ ложно,} \\ 1, & \text{если } P(x_1, \dots, x_n) \text{ истинно.} \end{cases}$$

Функция $\chi_p(x_1, \dots, x_n)$ рассматривается как функция от натуральных аргументов $x_1, \dots, x_n$, принимающая натуральные значения 0 или 1.

Пример. Предикаты $x_1 = x_2$, $x_1 < x_2$ являются п/р предикатами.

Свойство 1. Если $P(x_1, \dots, x_n)$, $Q(x_1, \dots, x_n)$ - п/р предикаты, то $\neg P$, $P \& Q$, $P \vee Q$, $P \oplus Q$, $P \sim Q$, $P \rightarrow Q$ тоже являются п/р предикатами.

Пример. Предикат $x_1 \leq x_2$, равный $(x_1 < x_2) \vee (x_1 = x_2)$, является п/р предикатом.

Свойство 2. Пусть п/р предикаты $P_1(x_1, \dots, x_n), \dots, P_k(x_1, \dots, x_n)$ таковы, что

$P_i \& P_j = 0$ тождественно при любых $i \neq j$,
 $P_1 \vee P_2 \vee \dots \vee P_k = 1$ тождественно, т.е. на любом наборе переменных $x_1, \dots, x_n$ равен 1 один и только один из предикатов $P_1, \dots, P_k$.
 Пусть также $g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n)$ - п/р функции.

Тогда функция $f(x_1, \dots, x_n) = \begin{cases} g_1(x_1, \dots, x_n), & \text{если } P_1(x_1, \dots, x_n) = 1 \\ g_k(x_1, \dots, x_n), & \text{если } P_k(x_1, \dots, x_n) = 1 \end{cases}$

является примитивно рекурсивной.

Функция f совпадает с одной из функций $g_1, \dots, g_k$ в зависимости от того, какой из предикатов $P_1, \dots, P_k$ равен 1. При $k = 2$ говорят, что функция f получается из функций g_1, g_2 применением **условного оператора**. Можно показать, что, например, функции $\min(x, y)$, $\max(x, y)$ - примитивно рекурсивны.

4. Оператор минимизации.

Свойство 1. Если $f(x_1, \dots, x_n, y)$ - п/р функция, то ограниченная сумма $g(x_1, \dots, x_n, a) = \sum_{y < a} f(x_1, \dots, x_n, y)$, при $f(x_1, \dots, x_n, 0) = 0$, тоже является п/р функцией.

Свойство 2. Если $f(x_1, \dots, x_n, y)$ - п/р функция, то ограниченное произведение $g(x_1, \dots, x_n, a) = \prod_{y < a} f(x_1, \dots, x_n, y)$, при $f(x_1, \dots, x_n, 0) = 1$, тоже является п/р функцией.

Сравни пример 3 п. 1.

Свойство 3. Если $P(x_1, \dots, x_n, y)$ - п/р предикат, то ограниченный квантор $Q(x_1, \dots, x_n, a) = \forall y_{y \leq a} P(x_1, \dots, x_n, y)$, при $P(x_1, \dots, x_n, 0) = 1$, тоже является п/р предикатом.

Свойство можно рассматривать как обобщение конъюнкции для конечного числа логических сомножителей.

Свойство 4. Если $P(x_1, \dots, x_n, y)$ - п/р предикат, то ограниченный квантор $Q(x_1, \dots, x_n, a) = \exists y_{y \leq a} P(x_1, \dots, x_n, y)$, при $P(x_1, \dots, x_n, 0) = 0$, тоже является п/р предикатом.

Свойство можно рассматривать как обобщение дизъюнкции для

конечного числа логических слагаемых.

Результатом применения **ограниченного оператора минимизации** (μ - оператора) к предикату $P(x_1, \dots, x_n, y)$ называется функция $f(x_1, \dots, x_n, z)$, равная такому значению $y \in \{0, 1, \dots, z\}$, при котором $P(x_1, \dots, x_n, y) = 1$ и $\forall t \leq y \ P(x_1, \dots, x_n, t) = 0$;

если же $P(x_1, \dots, x_n, y) = 0$ для всех $y \in \{0, 1, \dots, z\}$, то полагают $f(x_1, \dots, x_n, z) = z + 1$, чтобы функция f была всюду определенной. Обозначение: $f(x_1, \dots,$

$$x_n, z) = \mu_{y \leq z} P(x_1, \dots, x_n, y).$$

В частности, при $n = 0$ функция $\mu_{y \leq z} P(y)$ - это константа $c < z$ такая, что $P(c) = 1$ и $P(t) = 0$, если $t < c$.

Примеры. 1) $\mu_{y \leq 10} (y^2 + y > 6) = 3$

2) $\mu_{y \leq 10} (2^y > 100) = 7, \mu_{y \leq 4} (2^y > 100) = 5.$

Замечание. Обратите внимание на два последних равенства. Среди чисел $0, 1, \dots, 10$ есть такие, для которых $2^y > 100$, наименьшее среди них - 7; если же допустимое значение y не превосходит 4, то таких чисел нет, поэтому значение функции равно $4 + 1 = 5$.

При $n = 1$ формула $\mu_y P(x, y)$ выражает функцию $f(x)$ такую, что $P(x, f(x)) = 1$ и $P(x, f(t)) = 0$ для всех $t < x$.

Теорема 1. Если $P(x_1, \dots, x_n, y)$ - п/р предикат, то п/р функцией является и $\mu_{y \leq z} P(x_1, \dots, x_n, y)$.

Результатом применения **ограниченного оператора минимизации** (μ - оператора) к функции $g(x_1, \dots, x_n, y)$ называется функция $f(x_1, \dots, x_n, z)$, равная $\mu_{y \leq z} (g(x_1, \dots, x_n, y) = z)$.

Из теоремы 1 следует, что если $g(x_1, \dots, x_n, y)$ - п/р функция, то $\mu_{y \leq z} (g(x_1, \dots, x_n, y) = z)$ - тоже п/р функция.

Примеры. 1) Целочисленные функции $[\sqrt{x}]$, $[\log_k x]$, $[z/x]$, доопределенные при $x = 0$, являются п/р функциями. Действительно:

$$[\sqrt{x}] = \mu_y \left((y+1)^2 > x \right);$$

$$[\log_k x] = \mu_y (k^{y+1} > x);$$

$$[z/x] = \mu_y (x \cdot (y+1) > z).$$

2) Предикат $K(x, y) = X:Y$ ("x является делителем y", или "y делится на x") - п/р предикат, так как $X:Y = (x \cdot [y/x] = y)$.

3) Предикат $K_{\text{общ}}(x_1, x_2, y)$ - "y есть общее кратное чисел x_1, x_2 " является п/р, так как $K_{\text{общ}}(x_1, x_2, y) = K(x_1, y) \& K(x_2, y)$. Пусть теперь $f(x_1, x_2, z) = \mu_y K_{\text{общ}}(x_1, x_2, y)$. Если для наименьшего общего кратного чисел x_1, x_2 - НОК(x_1, x_2) - выполнено НОК(x_1, x_2) $< z$, то $f = \text{НОК}(x_1, x_2)$, иначе $f = z + 1$.

$$\text{Так, } f(15, 10, 100) = \mu_{y \leq 100} K_{\text{общ}}(15, 10, y) = 30;$$

$$f(15, 10, 20) = \mu_{y \leq 20} K_{\text{общ}}(15, 10, y) = 21,$$

так как среди чисел 0, 1, ..., 20 нет общих кратных чисел 10 и 15.

5. Обратимся к еще одной рекурсивной схеме.

Теорема 2. Если n -местные функции $g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n)$ и $(n+k+1)$ -местные функции $h_1(x_1, \dots, x_n, y, u_1, \dots, u_k), \dots, h_k(x_1, \dots, x_n, y, u_1, \dots, u_k)$ являются п/р, то и $(n+1)$ -местные функции $f_1, \dots, f_k$, определяемые по схеме совместной рекурсии следующим образом:

$$f_i(x_1, \dots, x_n, 0) = g_i(x_1, \dots, x_n) \text{ при } i = 1, 2, \dots, k,$$

$$f_i(x_1, \dots, x_n, y+1) = h_i(x_1, \dots, x_n, y, f_1(x_1, \dots, x_n, y), \dots, f_k(x_1, \dots, x_n, y)) \text{ при } i = 1, 2, \dots, k, \text{ также являются п/р.}$$

Можно рассмотреть такую интерпретацию примитивной рекурсии вообще и совместной рекурсии в частности. Схема из функциональных элементов и задержки на рис. 5.1 а описывает оператор с каноническим уравнением $f(x, t) = h(x(t), h(t-1))$, или по-другому, $f(x, t) = h(x(t), f(x, t-1))$. В начальный момент $t = 0$ значение на выходе задержки $z(0) = c$ (c - некоторая константа; для логических сетей мы полагали для единообразия, что $z(0) = 0$, но для детерминированности работы схемы это не обязательно, - достаточно, чтобы c было фиксировано).

Итак, для $t = 0$ имеем $f(x, 0) = h(x, c) = g(x)$. В момент $t = 1$ $f(x, 1) = h(x, f(x, 0))$; в общем случае $f(x, t+1) = h(x, f(x, t))$. Таким образом, схема

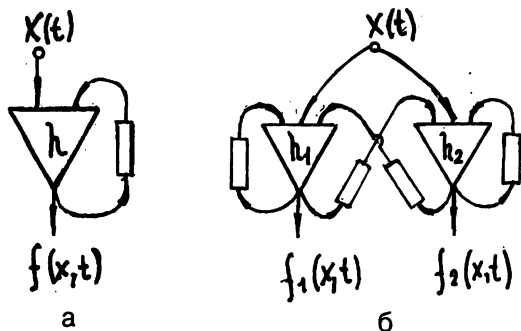


Рис. 5.1

рис. 5.1a реализует примитивную рекурсию по переменной t , т.е. по времени.

По аналогии можно считать, что схема рис. 5.1б реализует оператор совместной рекурсии по времени:

$$\begin{aligned} f_1 &= h_1(x(t), f_1(t-1), f_2(t-1)), \\ f_2 &= h_2(x(t), f_1(t-1), f_2(t-1)), \end{aligned}$$

или по-другому:

$$\begin{aligned} f_1(x, t+1) &= h_1(x, f_1(x, t), f_2(x, t)), \\ f_2(x, t+1) &= h_2(x, f_1(x, t), f_2(x, t)). \end{aligned}$$

По существу совместная рекурсия дает описание функции-вектора $(f_1, \dots, f_k)$. Если закодировать этот вектор натуральным числом, причем так, чтобы существовала однозначная и примитивно рекурсивная расшифровка этого кода (т.е. извлечение нужной компоненты вектора), то таким путем можно показать, что совместная рекурсия - тоже п/р. В качестве такого кода числового кортежа $\{d_1, d_2, d_3, \dots, d_n\}$ ($d_i = 0, 1, \dots$) можно использовать его так называемый **гёделевский номер** (по имени

немецкого математика **К. Гёделя**): число $K = 2^{d_1} \cdot 3^{d_2} \cdot 5^{d_3} \cdot \dots \cdot p_n^{d_n}$, где p_i -е по счету простое число ($p_1 = 2, p_2 = 3, p_3 = 5, \dots$). Тогда d_i - показатель при p_i в разложении числа K на простые множители, и можно показать, что вычисление d_i по K проводится п/р процедурой.

6. Частично рекурсивные функции.

Итак, из простейших - исходных - функций с помощью операторов суперпозиции и примитивной рекурсии можно получить огромное разнообразие функций, включая основные функции арифметики, алгебры и анализа (с поправкой на целочисленность). Поэтому все эти функции естественно считать вычислимыми.

Остановимся на вопросе о целочисленности примитивно рекурсивных функций. Можно убедиться, что это не является ограничением. Так, дробное число X/Y можно представить словом a/b , где a и b - коды числителя и знаменателя, а знак операции деления кодируется особым символом. Таким же образом можно, вводя специальное кодирование для символов арифметических процедур, записывать словами любые формулы, а их, в свою очередь, можно кодировать числами.

Например, десятичные цифры $0, \dots, 9$ кодируют себя сами, параметры и переменные, обозначаемые $b_1, b_2, \dots$ кодируются четными числами $10,$

12 и т.д. (код $b_n - 2n + 8$); символы арифметических действий – нечетными числами, начиная с 11, согласно нижеприведенному

$$\begin{array}{ccccccc}
 (\div 11 & - \div 17 & = + 23 & \sqrt[4]{} \div 29 \\
) \div 13 & * \div 19 & \sqrt{} \div 25 & \dots\dots\dots \\
 + \div 15 & / \div 21 & \sqrt[3]{} \div 27 & \sqrt{} \div (2r + 21)
 \end{array}$$

Например, формула $\sqrt[4]{(20 - b_1 + 7/3 * b_3)}$ будет представлена словом: 31, 11, 2, 0, 17, 10, 19, 19, 2, 15, 7, 21, 3, 19, 14, 13 (операция возведения в степень кодируется дублированием символа умножения **). Этому набору чисел может быть поставлен в соответствие его целочисленный гёделевский номер:

$$2^{31} \cdot 3^{11} \cdot 5^2 \cdot 7^0 \cdot 11^{17} \cdot 13^{10} \cdot 17^{19} \cdot 19^{19} \cdot 23^2 \cdot 29^{15} \cdot 31^{17} \cdot 37^{21} \cdot 41^3 \cdot 43^{19} \cdot 47^{14} \cdot 53^{13}.$$

Возникает вопрос, все ли функции, интуитивно причисляемые к вычислимым, являются п/р? Ответ - отрицательный: можно, например, построить функцию, которая растет быстрее, чем любая п/р функция. Отсюда можно заключить, что описанные средства построения вычисляемых функций нуждаются в расширении. Такое расширение достигается включением в число допустимых неограниченного μ -оператора. Однако при этом, в отличие от п/р, могут получиться не всюду определенные функции:

$$\text{если } f(x_1, \dots, x_n) = \mu y (g(x_1, \dots, x_n, y) = 0),$$

то, вычисляя значения g на наборах $(x_1, \dots, x_n, 0)$, $(x_1, \dots, x_n, 1)$ и т.д. до тех пор, пока не получим значения 0, мы вычислим значение функции f только в том случае, если такое y существует.

Результатом применения **неограниченного оператора минимизации (μ -оператора)** к предикату $P(x_1, \dots, x_n, y)$ называется функция $f(x_1, \dots, x_n)$, равная такому значению $y \in N^0$, при котором $P(x_1, \dots, x_n, y) = 1$ и $P(x_1, \dots, x_n, t) = 0$ для всех $t < y$; если же такого значения y нет, то $f(x_1, \dots, x_n)$ считается не определенной. Пишут $f(x_1, \dots, x_n) = \mu y P(x_1, \dots, x_n, y)$.

То же определение можно записать по-другому:

$$f(x_1, \dots, x_n) = \mu y P(x_1, \dots, x_n, y) = \min_{P(x_1, \dots, x_n, t)=1} \{0, 1, \dots, t, \dots, z\}$$

Результатом применения **неограниченного оператора минимизации (μ -оператора)** к функции $g(x_1, \dots, x_n, y)$ по переменной y

называется функция $f(x_1, \dots, x_n, z)$, равная $\mu y (g(x_1, \dots, x_n, y) = z)$.

То же в другой записи: $f(x_1, \dots, x_n, z) = \mu y g(x_1, \dots, x_n, y) =$

$$\min_{g(x_1, \dots, x_n, t)=z} \{0, 1, \dots, t, \dots, z\}$$

Примеры. 1) Функция $f(x) = \mu y (y + 1 = x)$ не определена при $x = 0$, т.е. эта функция - не п/р.

2) Функция $f(x) = \mu y (7 < x \cdot y \leq 10)$ определена только для значений $0 < x \leq 10$, кроме $x = 6, 7$. Ее значения приведены ниже:

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...
f(x)	-	8	4	3	2	2	-	-	1	1	1	-	-	-	...

Функция называется **частично рекурсивной (ч/р)**, если она может быть получена из исходных применением конечного числа операторов S (суперпозиции), R (примитивной рекурсии) и неограниченных μ -операторов.

Оказывается, что понятие частично рекурсивной функции является в некотором смысле алгоритмически достаточным.

Тезис Черча: всякая эффективно вычислимая функция является частично рекурсивной.

Смысл тезиса Черча и некоторые основания считать его справедливым изложены в п. 4 § 1.

Частично рекурсивная функция называется **общерекурсивной (о/р)**, если она определена всюду на $\mathbb{N}^0$.

Свойство 5. Всякая п/р функция является о/р.

П/р, ч/р и о/р функции называются **рекурсивными функциями (р. ф.)**.

7. *Универсальная общерекурсивная функция.*

Рассмотрим произвольную систему S частичных, т.е. не обязательно всюду определенных n-местных функций. Частичная функция $F(x_0, x_1, \dots, x_n)$ от $n + 1$ переменных называется **универсальной** для семейства S, если выполнены два условия:

а) для каждого фиксированного числа i n-местная функция $F(i, x_1, \dots, x_n)$ принадлежит S;

б) для каждой функции $f(x_1, \dots, x_n)$ из S существует такое число i, что для всех $x_1, \dots, x_n$ выполнено $F(i, x_1, \dots, x_n) = f(x_1, \dots, x_n)$.

Иначе говоря, функция $F(x_0, x_1, \dots, x_n)$ универсальна для семейства S, если все функции из S можно расположить в последовательность

$F(0, x_1, \dots, x_n), F(1, x_1, \dots, x_n), \dots, F(i, x_1, \dots, x_n), \dots$

Число i называется **номером** функции $F(i, x_1, \dots, x_n)$, а полученная таким образом нумерация функций семейства S называется нумерацией, отвечающей универсальной функции F . Обратно, если задана какая-либо нумерация системы S , т.е. если задано какое-либо отображение $i \rightarrow f_i$ натурального ряда на S , то функция $F(x_0, x_1, \dots, x_n)$, определенная формулой $F(x, x_1, \dots, x_n) = f_x(x_1, \dots, x_n)$, является универсальной для S .

Заметим, что рассматриваемые нумерации - не одно-однозначные: различные натуральные числа i, j могут быть номерами одной и той же функции из S .

Пусть теперь $F(x, x_1, x_2, \dots, x_n)$ - универсальная функция для семейства S некоторых всюду определенных функций от n переменных. Из определения следует, что эта универсальная функция будет также всюду определенной. Введем новую функцию g , полагая

$$g(x_1, x_2, \dots, x_n) = F(x_1, x_1, x_2, \dots, x_n) + 1. \quad (*)$$

Покажем, что функция $g(x_1, x_2, \dots, x_n)$ не входит в семейство S . Действительно, иначе для подходящего i выполнялось бы равенство $g(x_1, x_2, \dots, x_n) = F(i, x_1, x_2, \dots, x_n)$, и, следовательно,

$$F(i, x_1, x_2, \dots, x_n) = F(x_1, x_1, x_2, \dots, x_n) + 1.$$

Но при $x_1 = i$ это равенство обращается в противоречивое соотношение

$$F(i, i, x_2, \dots, x_n) = F(i, i, x_2, \dots, x_n) + 1,$$

что и доказывает наше утверждение.

Прием, примененный в приведенном доказательстве, носит название **диагонального метода (или процесса) Кантора**.

Отметим два важных следствия доказанного утверждения.

Пусть S - система всех n -местных примитивно рекурсивных (соответственно, общерекурсивных) функций. Равенство $(*)$ показывает, что если функция $F(x_0, x_1, x_2, \dots, x_n)$ п/р (соответственно, о/р), то таковой является и функция g (она получается из F подстановкой переменной x_1 вместо x_0 и прибавлением константы 1). Но функция g не входит в S и, значит, она не п/р (соответственно, не о/р). Тем самым получена простая, но существенная теорема.

Теорема 3. Система всех n -местных примитивно рекурсивных функций не имеет примитивно рекурсивной универсальной функции. Система всех n -местных общерекурсивных функций не имеет общерекурсивной универсальной функции ($n = 1, 2, \dots$).

В то же время справедлива и следующая теорема.

Теорема 4. Система всех одноместных примитивно рекурсивных функций имеет общерекурсивную универсальную функцию.

§3. Машины Тьюринга

1. Машина Тьюринга (МТ или м/т) задается тройкой $T = (A; Q, P)$, где $A = \{a_1, \dots, a_n\}$ - внешний алфавит, часто выбирают $A = \{0, 1\}$; $Q = \{q_0, q_1, \dots, q_m\}$ - внутренний алфавит, элементы которого обозначают состояния управляющего устройства, обычно q_0 - заключительное состояние, q_1 - начальное состояние; P - **программа**, состоящая из команд вида $q_i a_j \rightarrow q_k a_j S$, где S - один из символов сдвига: L (налево), R (направо), E (на месте); $1 \leq q_i \leq m$, $0 \leq q_k \leq m$, $a_j, a_j \in A$.

Пара символов до стрелки называется левой частью команды; тройка символов после стрелки - правой частью. Для каждого i, j программа имеет не более одной команды с левой частью $q_i a_j$.

Машина функционирует в дискретном времени $t = 0, 1, 2, \dots$ следующим образом.

Имеется бесконечная в обе стороны лента, разбитая на ячейки (т.е. множество ячеек, упорядоченное по типу Z - множества целых чисел). В начальный момент в конечном числе ячеек записаны символы из A , в остальных ячейках записан пустой символ, обозначаемый через 0 . Машина Тьюринга имеет читающую-пишущую головку, которая в любой момент дискретного времени обозревает некоторую ячейку (т.е. находится в ячейке и воспринимает написанный в ней символ) - рис. 5.2. МТ имеет управляющее устройство (автомат), которое в каждый момент времени находится в одном из состояний $q_i \in Q$; из Q выделяется подмножество Q_0 заключительных состояний, которое может быть и пустым; обычно $Q_0 = \{q_0\}$, т.е. Q содержит одно заключительное состояние.

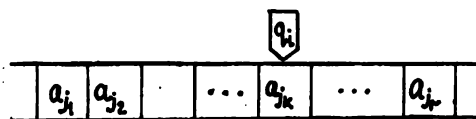


Рис. 5.2

Шаг работы МТ. Если МТ находится в состоянии q_i и в обозреваемой ячейке записан символ a_j , то выполняется (единственная!) команда с левой частью, образованной этой

парой: $q_i a_j \rightarrow q_k a_j S$; МТ переходит в состояние q_k , в ячейке на место a_j записывается a_j и головка сдвигается по ленте согласно S : либо на одну ячейку влево или вправо, либо остается на месте. Таким образом, совершается детерминированный переход: символ a_j воспринимается как входной, а a_j - как выходной для автомата. Затем совершается следующий детерминированный шаг по той же схеме.

В начальный момент МТ находится в состоянии q_1 ; если в процессе работы МТ переходит в одно из заключительных состояний, то она останавливается (команд, в левой части которых присутствует q_0 , в программе нет). МТ останавливается также, если отсутствует команда, которую нужно выполнить (т.е. с левой частью, соответствующей текущему полному состоянию МТ): это называется безрезультатной остановкой.

2. Конфигурацией (машинным словом) называется слово K вида Uq_1V , где U - слово на ленте левее головки, V - слово правее слова U , начиная с символа, находящегося в той же ячейке, где расположена головка. Стандартная начальная конфигурация – q_1V , а стандартная заключительная – q_0V . Это значит, что в начале работы головка находится в начальном состоянии в самой левой из заполненных ячеек; заполненные ячейки могут не составлять связного куска, а перемежаться незаполненными участками; заключительная конфигурация: головка в заключительном состоянии также находится в самой левой из заполненных ячеек.

Если при выполнении некоторой команды машины T из конфигурации K_1 получается конфигурация K_2 , то пишут $T:K_1 \rightarrow K_2$. Если $T:K_{i_1} \rightarrow K_{i_2} \rightarrow \dots \rightarrow K_{i_n}$, то пишут $T:K_{i_1} \Rightarrow K_{i_n}$. Наконец, если K_1 - начальная, а K_n - заключительная конфигурации, то пишут $T:K_1 \vdash K_n$, а если $T:K_1 \rightarrow K_2 \rightarrow \dots \rightarrow K_n \rightarrow \dots$ (т.е. машина T ни в какой момент времени не переходит в заключительное состояние), то пишут $T:K_1 \vdash$.

Детерминированная последовательность $K_{i_1}, K_{i_2}, \dots$ представляет **протокол работы машины T** . Важно подчеркнуть, что как в начальный, так и в каждый из последующих моментов работы МТ конфигурация на ленте остается конечной (слово в алфавите $A \cup Q$).

Говорят, что машина T **применима** к слову A , если T , начав работу на слове A , остановится через конечное число шагов; если при этом получается слово B , то пишут $T(A) = B$. Если T не останавливается, то T **неприменима** к слову A . В последнем случае совокупность всех конфигураций в процессе бесконечной работы может быть как ограниченной (тогда она периодична: ситуация **зацикливания**), так и неограниченной. М/т считается применимой к множеству слов $\{A_i\}$, если она применима к каждому слову A_i .

В следующих примерах 1-4 внешний алфавит $A = \{0, 1\}$, выражение a^x означает слово $aa\dots a$ - x раз; аналогично B^x , где B - слово, означает слово $BB\dots B$ - x раз.

Пример 1. М/т СЛ ("сдвиг слова 1^x влево"): $q_1001^x0 \vdash q_001^x0$ можно реализовать следующей программой:

	комментарий
$q_1 0 \rightarrow q_2 0R$	
$q_2 0 \rightarrow q_3 1R$	начало сдвига
$q_3 1 \rightarrow q_3 1R$	цикл движения головки вправо
$q_3 0 \rightarrow q_4 0L$	начало обратного хода
$q_4 1 \rightarrow q_5 0L$	конец сдвига
$q_5 1 \rightarrow q_5 1L$	цикл движения головки влево
$q_5 0 \rightarrow q_0 0E$	останов

Замечание. Обратите внимание, что команды с левыми частями $q_1 1$, $q_2 1$, $q_4 0$ в программе отсутствуют: это допустимо, поскольку $q_1 1$, $q_2 1$, $q_4 0$ - невозможные ситуации при данной начальной конфигурации.

Приведем протокол применения м/т СЛ к начальному слову 111, т.е. при $X = 3$. Жирным шрифтом выделены изменяющиеся символы:

```

... q1 0   0   1   1   1   0 ...
...   0 q2 0   1   1   1   0 ...
...   0   1 q3 1   1   1   0 ...
...   0   1   1 q3 1   1   0 ...
...   0   1   1   1 q3 1   0 ...
...   0   1   1   1   1 q3 0 ...
...   0   1   1   1   1 q4 1   0 ...
...   0   1   1 q5 1   0   0 ...
...   0   1 q5 1   1   0   0 ...
...   0 q5 1   1   1   0   0 ...
...   0 q0 1   1   1   0   0 ...

```

Можно записывать протокол в более компактном виде, заменяя фрагмент $q_i X$ обозначением X_i , т.е. конфигурация изображается словом во внешнем алфавите м/т, а символ X_i обозначает положение головки в состоянии q_i на ячейке с символом X . Кроме того, поскольку каждый шаг работы м/т изменяет только ячейки, ближайшие к головке, можно не повторять в каждой строке протокола всю конфигурацию, а приводить лишь символы в окрестности головки, выделяя жирным шрифтом изменяющийся символ. Остальные символы совпадают с находящимися в тех же ячейках и обозначенными в предыдущих строках. Отметим также, что слева и справа от слова на ленте располагаются пустые символы, которые обозначаются в том случае, если в ячейке с пустым символом находится головка м/т. Предыдущий протокол в этой форме выглядит так:

$q_3 0 \rightarrow q_4 R$	$q_3 1 \rightarrow R$
$q_4 0 \rightarrow q_5 1 L$	$q_4 1 \rightarrow R$
$q_5 0 \rightarrow q_6 L$	$q_5 1 \rightarrow L$
$q_6 0 \rightarrow q_2 1 R$	$q_6 1 \rightarrow L$
$q_7 0 \rightarrow q_0 E$	$q_7 1 \rightarrow R$

Упражнение. Запишите в сокращенном виде программы из примеров 1-3.

Программу м/т можно задавать также таблицей переходов и графом переходов подобно автомату (которым и является управляющее устройство м/т) - надо лишь указывать, кроме выходного символа, символ сдвига. Таблица переходов м/т представляет собой табличное задание программы м/т: строки соответствуют состояниям управляющего устройства, а столбцы - символам внешнего алфавита; в клетках таблицы в полном или сокращенном виде записываются правые части соответствующих команд м/т. Пример предыдущей программы K1 - в таблице 5.1 и на рис. 5.3.

Таблица 5.1

	0	1
q_1	$q_2 R$	
q_2	$q_7 L$	$q_3 0 R$
q_3	$q_4 R$	R
q_4	$q_5 1 L$	R
q_5	$q_6 L$	L
q_6	$q_2 1 R$	L
q_7	$q_0 E$	R

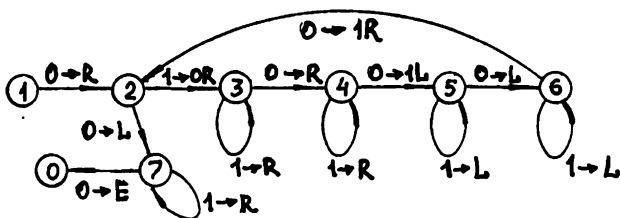


Рис. 5.3

В следующих примерах 5-8 обрабатываются слова $P(a, b)$ в алфавите $\{a, b\}$; при этом используются дополнительные (служебные) символы-метки $c, d, *$. Результат в примере 6 - слово в алфавите $\{1\}$.

Пример 5. Рассмотрим два протокола применения м/т Т с программой $q_1 a \rightarrow q_2 b R; q_1 b \rightarrow q_2 a R; q_2 a \rightarrow q_0; q_2 b \rightarrow q_1 L$.

а) Начальная конфигурация q_1aa :

0	a_1	a	0
0	b	a_2	0
0	b	a_0	0

Результат: $aa \vdash ba$.

б) Начальная конфигурация q_1ab :

0	a_1	b	0
0	b	b_2	0
0	b_1	$\bar{b}$	0
0	a	b_2	0
0	a_1	b	0

.....

Пятая конфигурация протокола совпадает с первой (головка в состоянии q_1 на символе a слова ab). Поскольку работа м/т детерминирована, следующая, шестая конфигурация совпадает со второй и т.д. Мы имеем зацикливание, т.е. м/т Т неприменима к начальной конфигурации q_1ab . Результат: $ab \vdash$.

Пример 6. М/т N_a ("число символов a"). Программа – в таблице 5.2. Начальная конфигурация - головка слева от слова $P(a, b) : q_10P(a, b)0 \vdash q_01^n0$. М/т подсчитывает n - число символов a в слове $P(a, b)$ и записывает результат словом 1^n в алфавите $\{1\}$. Внешний алфавит $A = \{0, 1, a, b, c\}$, c - вспомогательный символ.

Таблица 5.2

	0	1	a	b	c
q_1	q_5R				
q_2	q_31L	R	R	R	
q_3	q_0	L	L	L	q_5R
q_4	R	q_0L	$0L$	$0L$	$0L$
q_5	q_3L	q_4L	q_2cR	R	

Упражнение. Постройте протокол работы м/т N_a для начального слова

- а) abbaaba
- б) bbabaab
- в) aaababa
- г) aaaaa
- д) bbbbbb.

Пример 7. М/т К2 ("дублирование") перерабатывает слово $P(a, b)$ в слово $P^2 = PP$: $q_1 P(a, b) \vdash q_0 P(a, b) P(a, b)$. Программа – в таблице 5.3. Начальная конфигурация – головка в ячейке, содержащей левый символ слова $P(a, b)$. Внешний алфавит $A = \{0, a, b, c, d\}$; c, d – вспомогательные символы.

Таблица 5.3

	a	b	c	d	0
q_1	$q_4 c R$	$q_5 d R$	R	R	$q_6 L$
q_2	q_3	q_3	L	L	
q_3	L	L	$q_1 R$	$q_1 R$	
q_4	R	R			$q_2 c L$
q_5	R	R			$q_2 d L$
q_6			$a L$	$b L$	$q_0 R$

Пример 8. М/т Рфл ("отражение") перерабатывает слово $P(a, b)^*$, т.е. слово $P(a, b)$, дополненное справа разделителем – символом $*$, в слово $P^* P^{-1}$, где P^{-1} – запись слова P в обратном порядке: $q_1 P(a, b)^* \vdash q_0 P(a, b)^* P^{-1}(a, b)$. Программа – в таблице 5.4. Начальная конфигурация – головка в ячейке, содержащей левый символ слова $P(a, b)$. Внешний алфавит $A = \{0, a, b, c, d, *\}$; c, d – вспомогательные символы.

Таблица 5.4

	a	b	c	d	*	0
q_1	$q_2 c R$	$q_3 d R$			$q_6 L$	q_0
q_2	R	R			$q_4 R$	
q_3	R	R			$q_5 R$	
q_4	$a R$	$q_5 a R$				$q_6 a L$
q_5	$q_4 b R$	$b R$				$q_6 b L$
q_6	L	L	$q_1 a R$	$q_1 b R$	L	q_0

Упражнение. Постройте протоколы работы м/т К2 и Рфл для начальных слов:

- а) abb
- б) $bbab$
- в) $aaab$

Упражнение. Постройте графы переходов м/т К2 и Рфл.

Приведенные примеры демонстрируют отличие м/т от конечного автомата: наличие ленты, на которой могут записываться слова

неограниченной длины, обеспечивает потенциально неограниченную внешнюю память м/т. Это значительно расширяет возможности м/т по сравнению с конечным автоматом. Так, преобразования входной информации машинами K1, K2, Pfl в примерах 4, 7, 8 не могут быть выполнены автоматом из-за конечной памяти последнего. Ниже рассматриваются некоторые возможности м/т.

3. Вычисление функций машинами Тьюринга.

Будем представлять число m как 01^m0 , а кортеж чисел $(m_1, \dots, m_n)$ как $01^{m_1}01^{m_2}0 \dots 01^{m_n}0$.

Говорят, что машина T **правильно вычисляет** функцию $f(x_1, \dots, x_n)$ с областью определения $D \in N^0 \times N^0 \times \dots \times N^0$ (сокращенно $f(\vec{X})$), если для $\vec{X} \in D$ верно, что $T : q_1 01^{x_1} 01^{x_2} 0 \dots 01^{x_n} 0 \vdash q_0 01^{f(\vec{X})} 0$, а для $X \notin D$ верно, что $T : q_1 01^{x_1} 01^{x_2} 0 \dots 01^{x_n} 0 \not\vdash$, т.е. неприменима.

В частности, для функций одной переменной X исходное слово $q_1 01^X 0$ перерабатывается в слово $q_0 01^{f(X)} 0$.

Функция называется **правильно вычислимой**, если есть МТ, которая ее правильно вычисляет.

Пример 1. Функция $Z(x)$ (т.е. константа 0) правильно вычислима.

М/т $Z : q_1 01^X 0 \vdash q_0 00$; программа:

$$\begin{array}{ll} q_1 0 \rightarrow q_2 1R & \\ q_2 1 \rightarrow q_2 0R & q_2 0 \rightarrow q_3 0L \\ q_3 1 \rightarrow q_0 0E & q_3 0 \rightarrow q_3 0L. \end{array}$$

Пример 2. Функция $N(x)$ ("next") правильно вычислима.

М/т $N : q_1 01^X 0 \vdash q_0 01^{X+1} 0$; программа:

$$\begin{array}{ll} q_1 0 \rightarrow q_2 0R & \\ q_2 0 \rightarrow q_3 1L & q_2 1 \rightarrow q_2 1R \\ q_3 0 \rightarrow q_0 0E & q_3 1 \rightarrow q_3 1L. \end{array}$$

Опишем вкратце, как построить м/т для сложения и умножения двух чисел. При сложении слово $q_1 01^X 01^Y 0$ требуется переработать в $q_0 01^{X+Y} 0$. Для этого достаточно сдвинуть слово 1^Y на одну позицию влево, т.е. заменить разделитель 0 на 1 и удалить самую правую единицу.

Если к слову $01^X 0$ (опускаем символ, обозначающий положение головки) применить копирование, а затем сложить полученные два одинаковых числа, то получится удвоение числа X . Его можно рассматривать как прибавление к числу X слагаемого X . Если теперь задана пара чисел $01^X 01^Y 0$, и прибавлять X к первому числу Y раз,

уменьшая каждый раз число Y на 1, то в результате получим произведение чисел X и Y .

Говорят, что предикат $P(x_1, \dots, x_n)$ является вычислимым, если его характеристическая функция $\chi_p(x_1, \dots, x_n)$ правильно вычислима.

4. Операции над машинами Тьюринга.

Пусть $T_1 = (A, Q_1, P_1)$, $T_2 = (A, Q_2, P_2)$ - две м/т с общим алфавитом A ; $Q_1 = \{q'_0, q'_1, \dots, q'_r\}$, $Q_2 = \{q''_0, q''_1, \dots, q''_s\}$. **Композицией машин T_1 и T_2** называется м/т $T_1T_2 = (A, Q, P)$; $Q = \{q''_0, q'_1, \dots, q'_r, q''_1, \dots, q''_s\}$, $P = P'_1 \cup P_2$, где P'_1 получается из P_1 заменой q'_0 на q''_1 (т.е. вместо окончания работы T_1 начинает работать T_2).

Например, композиция м/т Рфл (пример 7) и м/т Del (таблица 5.5) перерабатывает слово $P(a, b)^*$ в слово $*P^{-1}(a, b)$.

Таблица 5.5

	a	b	*	0
q_1				q_2R
q_2	0 R	0 R	q_0R	

Различные композиции машин Тьюринга позволяют реализовать комбинации функциональных операторов. Этим устанавливается связь рекурсивных функций и машин Тьюринга.

Свойство 1. Если $T_1: q_1K_1 \vdash q_0K_2$, а $T_2: q_1K_2 \vdash q_0K_3$, то $T_1T_2: q_1K_1 \vdash q_0K_3$. Это означает, что композиция машин последовательно выполняет работу первой и второй.

Замечание. Если $T_1: q_1K_1 \vdash q_0K_2$, но $T_2: q_1K_2 \vdash$, то $T_1T_2: q_1K_1 \vdash$; если $T_1: q_1K_1 \vdash$, то сразу $T_1T_2: q_1K_1 \vdash$ независимо от T_2 (т.е. неприменимость на любом из двух этапов влечет неприменимость композиции).

Из этого, в частности, следует следующее свойство.

Свойство 2. Если м/т T_1 правильно вычисляет функцию $f: N^0 \vdash N^0$, м/т T_2 правильно вычисляет функцию $g: N^0 \vdash N^0$, то композиция T_1T_2 правильно вычисляет суперпозицию $g(f)$ этих функций.

Теорема 1. Если функции $f, g_1, g_2, \dots, g_n$ правильно вычислимы, то и функция $h = S(f, g_1, g_2, \dots, g_n)$ правильно вычислима.

Теорема 2. Пусть функция $f(x)$ задана оператором рекурсии R : $f(0) = c$, $f(x+1) = h(x, f(x))$. Тогда, если $h(x, y)$ правильно вычислима, то и $f(x)$ правильно вычислима.

Теорема 3. Если $P(x_1, \dots, x_n, y)$ - вычислимый предикат, то функция $f(x_1, \dots, x_n) = \mu y P(x_1, \dots, x_n, y)$ правильно вычислима.

Поскольку правильно вычислимы исходные п/р функции и реализуемы машиной Тьюринга операторы S , R и μ , то справедлива теорема 4.

Теорема 4 (теорема Тьюринга). Все частично рекурсивные функции правильно вычислимы.

Верно и обратное.

Теорема 5. Всякая функция, вычисляемая машиной Тьюринга, частично рекурсивна.

Это утверждение может показаться неожиданным: машина Тьюринга осуществляет преобразования как числовых, так и нечисловых объектов, в то время как рекурсивная функция - чисто числовое образование. Однако всякую программу м/т можно представить набором чисел, поскольку символы алфавитов A и Q , а также слова в алфавите $A \cup Q \cup \{S\} \cup \{\rightarrow\}$, выражающие команды программы м/т, можно закодировать в цифровом алфавите и воспринимать как числа. После этого каждый шаг алгоритма может быть интерпретирован как ч/р преобразование данного набора чисел.

Теоремы 4 и 5 означают, что класс ч/р функций совпадает с классом правильно вычисляемых функций.

5. Универсальная машина Тьюринга.

Систему команд м/т можно воспринимать как описание действий конкретного автомата с данным алфавитом и индивидуальной программой, способного преобразовывать исходные данные на ленте, а также как текст - совокупность предписаний, которые может выполнять аккуратный вычислитель. Их проведение представляет собой алгоритмическую процедуру, состоящую в следующем.

Для текущей конфигурации $\alpha_1 a_m q_i a_j \alpha_2$ (a_m, a_j - символы внешнего алфавита A ; α_1, α_2 - слова в алфавите A , q_i - символ внутреннего алфавита Q) найти в системе команд команду с левой частью $q_i a_j$. Если правая часть этой команды имеет вид $q_k a_l R$, то заменить в текущей конфигурации подслово $q_i a_j$ на $a_l q_k$ (получится конфигурация $\alpha_1 a_m a_l q_k \alpha_2$); если правая часть имеет вид $q_k a_l L$, то заменить $a_m q_i a_j$ на $q_k a_m a_l$; если правая часть имеет вид $q_k a_l E$, то заменить $q_i a_j$ на $q_k a_l$.

Эти правила можно облечь в форму программы для машины Тьюринга U , которая по заданным на ленте программе любой m/t T и некоторой исходной ситуации X машины T выполняла бы преобразование X в результат применения T к слову X .

Такая m/t U называется **универсальной машиной Тьюринга**, которую можно также назвать **интерпретатором**.

Данное описание не вполне корректно и нуждается в уточнении, т.к. если мы говорим о любой m/t T , то можем столкнуться с тем, что внешний и внутренний алфавиты машины U не объемлют алфавитов интерпретируемой машины T (последние могут, например, содержать больше символов, чем U). Выход заключается в том, что можно кодировать символы A_T и Q_T словами в алфавите A_U . С использованием этого приема удастся, например, построить универсальную машину Тьюринга с 5 символами и 6 состояниями, т.е. содержащую 30 команд.

В терминах современных компьютеров можно считать, что именно универсальная m/t представляет собой вычислительную машину, а отдельные m/t - программы для универсальной m/t . Важно подчеркнуть, что идея универсального алгоритмического устройства не связана с технической стороной вопроса, а является математическим объектом, к тому же опирающимся на крайне малое число элементарных исходных понятий. Небезынтересно также, что основополагающие работы по теории алгоритмов (и, в частности, работы А. М. Тьюринга) выполнены в 30-х годах XX в., до создания ЭВМ.

Математическая практика и отмеченная выше связь с другими понятиями алгоритма приводят к утвердительному ответу на вопрос, сформулированный в §1: для всех ли процедур, алгоритмических в интуитивном смысле, существует реализующая их машина Тьюринга?

Тезис Тьюринга. Всякий алгоритм может быть реализован машиной Тьюринга.

Теоремы 4 и 5 §3 связывают тезис Тьюринга с тезисом Черча.

Касаясь прикладной стороны изложенных элементов теории алгоритмов, можно отметить, что как раз в силу инвариантности основных ее результатов прикладное значение теории не связано с тем, насколько близки практике те или иные модели алгоритмов. Машина Тьюринга весьма далека от современных компьютеров, а рекурсивные функции и другие, не рассматривавшиеся здесь модели алгоритмов (нормальные алгоритмы А.А.Маркова и другие) - от реальных языков программирования прежде всего из-за предельной скромности используемых средств. Однако именно простота средств, во-первых, делает эти модели удобным языком доказательств, а во-вторых,

позволяет понять, без чего обойтись нельзя, а без чего можно и какой ценой, т.е. отличать удобства применения от принципиальных возможностей. Иначе говоря, прикладное значение рассмотренных моделей заключается в том, что с их помощью удобно строить теорию, верную для любых алгоритмических моделей, в том числе и как угодно близких к практике.

§4. Понятие формальной грамматики

Пусть задан алфавит V и тем самым множество V^* всех конечных слов, или цепочек, в алфавите V . Формальный язык L в алфавите V – это произвольное подмножество слов $L \subseteq V^*$. Конструктивное описание формальных языков осуществляется с помощью формальных систем специального вида, называемых **порождающими грамматиками**.

Формальной порождающей грамматикой G называется тройка (T, W, P) , где $T = \{a_1, a_2, \dots, a_n\}$ – алфавит **терминальных символов**, которые принято обозначать малыми латинскими буквами;

$W = \{I, V_1, V_2, \dots, V_m\}$ – алфавит служебных, **нетерминальных символов**, обозначаемых большими латинскими буквами; I называется **начальным символом**; предполагается, что $T \cap W = \emptyset$;

P – конечное множество порождающих правил вида $\varepsilon \rightarrow \eta$, где ε, η – слова (цепочки) в объединенном алфавите $T \cup W$.

Говорят, что цепочка β непосредственно выводится из цепочки α (обозначение: $\alpha \rightarrow \beta$), если $\alpha = \gamma\varepsilon\delta$, $\beta = \gamma\eta\delta$ и $\varepsilon \rightarrow \eta \in P$, т.е. если β получается заменой в слове α некоторого подслоа ε подсловом η при наличии порождающего правила $\varepsilon \rightarrow \eta$. Говорят, что β выводится из α (обозначение: $\alpha \rightarrow^* \beta$), если существует последовательность цепочек $v_1, v_2, \dots, v_n$ такая, что $\alpha \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow \beta$, т.е. существует порождающий процесс, переводящий слово α в слово β порождающими правилами из P . Последовательность $\alpha, v_1, v_2, \dots, v_n, \beta$ называется **выводом β из α** . Множество всех цепочек в алфавите A обозначим через A^* , пустую цепочку – через ε , а множество непустых цепочек – через A^+ .

Формальным языком $L(G)$, порожденным грамматикой G , называется множество слов $\{\alpha: \alpha \in T, I \rightarrow^* \alpha\}$, т.е. множество цепочек, которые могут быть получены применением порождающих правил P , отправляясь от начального символа I .

В зависимости от структуры системы порождающих правил выделяют 4 типа грамматик и, соответственно, языков.

Грамматикой типа 0 называется грамматика, правила вывода которой имеют самый общий вид $\varepsilon \rightarrow \eta$, где $\varepsilon \in (T \cup W)^+$, $\eta \in (T \cup W)^*$.

Грамматикой типа 1 (контекстной грамматикой) называется грамматика, правила вывода которой имеют вид $\alpha A \beta \rightarrow \alpha \omega \beta$, где $\omega \in (T \cup W)^+$; цепочки α и β - это контекст правила. Они не изменяются при его применении.

Грамматикой типа 2 (контекстно свободной, или КС грамматикой) называется грамматика, правила вывода которой имеют вид $A \rightarrow \alpha$, где $\alpha \in (T \cup W)^*$.

Грамматикой типа 3 (регулярной грамматикой) называется грамматика, правила вывода которой имеют вид либо $A \rightarrow aB$, либо $A \rightarrow a$.

Тип формального языка определяется типом порождающей его формальной грамматики. Для $i = 0, 1, 2$ имеются примеры языков типа i , для которых доказано, что они не являются языками типа $(i + 1)$.

Пример 1. $L = \{a^{2n-1}\}$ - регулярный язык (тип 3). Он порождается регулярной грамматикой $G = (T, W, P)$, где $T = \{a\}$, $W = \{I, A\}$, $P = \{I \rightarrow a, I \rightarrow aAI\}$.

Пример 2. $L = \{a^n b^n\}$ - КС язык (тип 2). Он порождается КС грамматикой $G = (T, W, P)$, где $T = \{a, b\}$, $W = \{I\}$, $P = \{I \rightarrow ab, I \rightarrow aIb\}$.

Пример 3. $L = \{a^n b^n a^n\}$ - контекстный язык (тип 1). Он порождается контекстной грамматикой $G = (T, W, P)$, где $T = \{a, b\}$, $W = \{I, A, B, C\}$, $P = \{I \rightarrow ABA, B \rightarrow b, b \rightarrow ABCA, bC \rightarrow bb, AC \rightarrow DC, DC \rightarrow DA, DA \rightarrow CA, A \rightarrow a\}$.

Пример 4. Язык булевых формул с переменными a, b, c порождается КС грамматикой, где $T = \{a, b, c, \vee, \&, \neg, (,)\}$ (символы переменных, символы трех булевых операций, открывающая и закрывающая скобки); $W = \{I\}$; P содержит следующие правила:

$$\begin{aligned} I &\rightarrow (I \vee I) \\ I &\rightarrow (I \& I) \\ I &\rightarrow \neg I \\ I &\rightarrow a \end{aligned}$$

$$I \rightarrow b$$

$$I \rightarrow c.$$

Этот язык отличается от языка формул исчисления высказываний отсутствием символа импликации, которую, как известно, можно выразить через конъюнкцию, дизъюнкцию и отрицание. Число переменных можно увеличить: соответственно увеличится число правил.

Пример 5. Если в предыдущем языке три правила, вводящие операции $\vee$, $\&$, $\neg$, заменить четырьмя правилами, вводящими операции $+$, $-$, $*$, $/$, то получим язык арифметических выражений. Он отличается от обычного языка арифметических формул тем, что не учитывается ассоциативность сложения и умножения, и поэтому выражения содержат много скобок (то же можно сказать и о языке примера 4). Для полного совпадения с языком арифметических выражений в эту грамматику надо добавить правила, порождающие числовые константы, и произвольные обозначения переменных.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

- 1. Составьте логическую схему базы знаний по теме юниты.**

Для выполнения заданий 2 -9 необходимо определить и записать в таблице K1 значения булевых переменных $a_1 - a_{48}$ (нули и единицы), исходя из следующих параметров:

H - порядковый номер студента в списке группы;

Φ - число букв в фамилии студента.

Впишите свои параметры в табличку:

$H =$	$\Phi =$
-------	----------

Значения $a_1 - a_{48}$ выбираются из таблицы K2 по следующему правилу:

$a_1 - a_6$ - 6 чисел подряд, начиная с позиции H ;

$a_7 - a_{16}$ - 10 чисел подряд, начиная с позиции $\Phi + 20$;

$a_{17} - a_{26}$ - 10 чисел подряд, начиная с позиции $H + \Phi + 20$;

$a_{27} - a_{36}$ - 10 чисел подряд, начиная с позиции $H + 2\Phi + 40$;

$a_{37} - a_{48}$ - 12 чисел подряд, начиная с позиции $H + 3\Phi + 60$.

Таблица K1

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}
a_{17}	a_{18}	a_{19}	a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27}	a_{28}	a_{29}	a_{30}	a_{31}	a_{32}
a_{33}	a_{34}	a_{35}	a_{36}	a_{37}	a_{38}	a_{39}	a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}	a_{47}	a_{48}

Таблица K2

	1	5	10	15	20
	0 1 0 0 0	1 0 0 1 1	0 1 0 0 1	0 0 1 0 0	
20 +	0 0 1 0 0	1 0 1 1 1	1 0 0 1 1	0 0 0 1 1	
40 +	0 0 0 0 0	1 0 0 0 0	0 0 1 1 1	0 0 0 1 0	
60 +	0 1 0 0 1	1 0 1 0 1	1 0 1 1 0	0 1 1 0 1	
80 +	0 1 0 0 0	0 0 1 1 0	0 1 1 1 0	1 1 1 0 1	
100 +	0 1 1 1 0	0 0 1 0 0	1 1 0 0 1	1 0 1 0 0	
120 +	1 0 1 0 0	1 0 1 1 0	1 0 0 0 1	1 0 0 0 1	

2. Символами $a_1 - a_{28}$ заполнить квадратную матрицу 7×7 (диагональные и наддиагональные элементы). Продолжить полученную треугольную матрицу симметричным образом.

a_1	a_2	a_3	a_4	a_5	a_6	a_7
	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}
		a_{14}	a_{15}	a_{16}	a_{17}	a_{18}
			a_{19}	a_{20}	a_{21}	a_{22}
				a_{23}	a_{24}	a_{25}
					a_{26}	a_{27}
						a_{28}

1) Рассматривая полученную матрицу как матрицу соседства вершин неориентированного графа G с вершинами 1, 2, 3, 4, 5, 6, 7, построить изображение графа. (Рекомендация: расположите вершины графа в вершинах правильного семиугольника).

2) Построить матрицу инцидентий графа G .

3) Удалив петли (если они есть), найти цикломатическое число полученного графа.

3. Символами $a_{11} - a_{38}$ заполнить наддиагональные элементы квадратной матрицы 8×8 (диагональные элементы - нулевые). Продолжить полученную треугольную матрицу симметричным образом.

0	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}
	0	a_{18}	a_{19}	a_{20}	a_{21}	a_{22}	a_{23}
		0	a_{24}	a_{25}	a_{26}	a_{27}	a_{28}
			0	a_{29}	a_{30}	a_{31}	a_{32}
				0	a_{33}	a_{34}	a_{35}
					0	a_{36}	a_{37}
						0	a_{38}
							0

1) Рассматривая полученную матрицу как матрицу соседства вершин неориентированного графа G с вершинами 1 - 8, построить изображение графа G .

2) Определить цикломатическое число графа G .

3) Выделить какой-либо остов графа G .

4) Построить базис циклов графа G .

4. Заполнить таблицу КЗ следующим образом: из символов $a_{21} - a_{48}$ образовать 7 четырехзначных двоичных чисел $b_1 - b_7$.

$$b_1 = a_{21}a_{22}a_{23}a_{24}, \quad b_2 = a_{25}a_{26}a_{27}a_{28}, \quad b_3 = a_{29}a_{30}a_{31}a_{32},$$

$$b_4 = a_{33}a_{34}a_{35}a_{36}, \quad b_5 = a_{37}a_{38}a_{39}a_{40}, \quad b_6 = a_{41}a_{42}a_{43}a_{44},$$

$$b_7 = a_{45}a_{46}a_{47}a_{48}$$

Таблица К3

b_i в двоичной записи	b_i в десятичной записи	c_i	$d_i = c_i / L$ $i = 1, 2, \dots, 6$	Сообщения и их коды Фано
$b_1:$			$d_1:$	$A_1:$
$b_2:$			$d_2:$	$A_2:$
$b_3:$			$d_3:$	$A_3:$
$b_4:$			$d_4:$	$A_4:$
$b_5:$			$d_5:$	$A_5:$
$b_6:$			$d_6:$	$A_6:$
$b_7:$			$d_7:$	$A_7:$

$L =$

Перевести их в десятичную систему, прибавить к каждому 1. Получим числа $c_1, \dots, c_7$: $c_i = b_i + 1$. Вычислить сумму: $L = \sum_{i=1}^7 c_i$. Определить с точностью до 0.01 числа d_i : $d_i = c_i / L$, $i = 1, 2, \dots, 6$; вычислить

$$d_7 = 1 - \sum_{i=1}^6 d_i \quad (\text{тем самым сумма чисел } d_1, \dots, d_7 \text{ равна } 1).$$

Считая числа $d_1 - d_7$ частотами сообщений $A_1 - A_7$, построить код Фано для системы сообщений $A_1 - A_7$; занести результат в последний столбец таблицы К3.

5. Для сообщения $X = a_3 a_4 a_5 a_6$ построить код Хэмминга (сообщение X'). Двоичное число $a_7 a_8 a_9$ перевести в десятичную систему, прибавить 1. Полученное число i определяет номер разряда в сообщении X' , в котором следует произвести одиночную ошибку замещения: $X' \rightarrow X''$. Провести декодирование сообщения X'' по методу Хэмминга.

6. Четырехзначное двоичное число $M = a_2 a_3 a_4 a_5$ перевести в десятичную систему и прибавить 33. Полученное число N определяет начальное число спичек в игре НИМ, причем разрешается убирать со стола за один ход 1, 2, 3 или 4 спички).

1) Определить, какая из сторон (начинающая А или противоположная В) выигрывает при правильной игре.

2) Вычислить и поместить в таблицу К4 значения $M, N, b_1, b_2, b_3, \dots$, где b_i - увеличенное на 1 двузначное двоичное число, образованное символами $a_{2i} + 4a_{2i+1} + 5$ ($b_1 = a_6 a_7 + 1$, $b_2 = a_8 a_9 + 1$, $b_3 = a_{10} a_{11} + 1$ и т.д.).

Таблица K4

(M)	(N)	(b ₁) (b ₂) (b ₃)...
Выигрывающие ходы:		

Определить правильные ходы за выигрывающую сторону, если последовательность ходов другой стороны $\sim b_1, b_2, b_3, \dots$. Ходы из $\{b_i\}$ выбираются до тех пор, пока игра не закончена.

7. Перевести в десятичную систему двоичное число $a_{39}a_{40}a_{41}$ и прибавить 1. Полученное число i определяет номер программы машины Тьюринга.

а) Построить граф переходов выбранной машины Тьюринга T_i .

б) Перевести слово $R = a_{42}a_{43}a_{44}a_{45}a_{46}a_{47}$ в алфавите $\{0, 1\}$ в слово S в алфавите $\{a, b\}$, заменив 0 на a , 1 на b ; заполнить таблицу K5.

в) Применить машину Тьюринга T_i к начальной конфигурации q_1S . Составить протокол. Показать заключительную конфигурацию.

Таблица K5

$a_{39}a_{40}a_{41}$	i	$R = a_{42}a_{43}a_{44}a_{45}a_{46}a_{47}$	S

T_1 :

	a	b	c	0
q_1	$q_2 R$	R	R	$q_4 L$
q_2	q_1	$q_3 c L$	q_1	$q_4 L$
q_3	$q_3 b R$		$q_1 R$	
q_4	L	L	L	$q_0 R$

T_2 :

	a	b	c	0
q_1	$q_2 R$	R	R	$q_3 L$
q_2	q_1	$q_1 c L$	q_1	$q_3 L$
q_3	L	L	L	$q_0 R$

T_3 :

	a	b	c	0
q_1	$q_2 R$	R		$q_5 L$
q_2	q_1	$q_3 a L$	q_1	$q_5 L$
q_3	$q_4 c R$			
q_4	$q_1 R$			
q_5	L	L	L	$q_0 R$

T₄:

	a	b	c	0
q ₁	q ₂ R	R		q ₅ L
q ₂	q ₁	q ₃ a L	q ₁	q ₅ L
q ₃	q ₄ b R		q ₁ R	
q ₄	q ₁ R			
q ₅	L	L	L	q ₀ R

T₅:

	a	b	c	0
q ₁	R	q ₂ R	R	q ₃ L
q ₂	q ₁ c L	q ₁	q ₁	q ₃ L
q ₃	L	L	L	q ₀ R

T₆:

	a	b	c	0
q ₁	R	q ₂ R		q ₅ L
q ₂	q ₃ b L	q ₁	q ₁	q ₅ L
q ₃		q ₄ a R		
q ₄		q ₁ R		
q ₅	L	L	L	q ₀ R

T₇:

	a	b	c	0
q ₁	R	q ₂ R	R	q ₄ L
q ₂	q ₃ c L	q ₁	q ₁	q ₄ L
q ₃		q ₃ a R	q ₁ R	
q ₄	L	L	L	q ₀ R

T₈:

	a	b	c	0
q ₁	R	q ₂ R		q ₅ L
q ₂	q ₃ b L	q ₁	q ₁	q ₅ L
q ₃		q ₄ c R		
q ₄		q ₁ R		
q ₅	L	L	L	q ₀ R

8. Двоичное число $a_{34}a_{35}a_{36}$ определяет набор функциональных элементов А, В, С логической сети (рис. 1) согласно таблице К6.

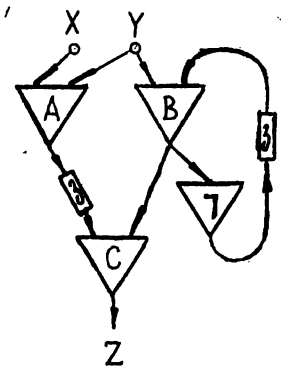


Рис.1

Таблица К6

	A	B	C
0 0 0	V	&	⊕
0 0 1	V	⊕	&
0 1 0	&	V	⊕
0 1 1	&	⊕	V
1 0 0	⊕	V	&
1 0 1	V	⊕	&
1 1 0	⊕	&	V
1 1 1	&	⊕	V

Двоичные числа $a_{37}a_{38}a_{39}a_{40}a_{41}a_{42}$ и $a_{43}a_{44}a_{45}a_{46}a_{47}a_{48}$ представляют входные последовательности X(t) и Y(t) (t = 1, 2, ..., 6). Определить выходную последовательность Z(1), Z(2), Z(3), Z(4), Z(5), Z(6).

Исходные данные и результат поместить в таблицу К7.

Таблица К7

$a_{34}a_{35}a_{36}$	$X(t) = a_{37}a_{38}a_{39}a_{40}a_{41}a_{42}$
A B C	$Y(t) = a_{43}a_{44}a_{45}a_{46}a_{47}a_{48}$
	Z(t)=

Указание. Составить формулу, выражающую оператор $X(t) = (X(t-1) \text{ A } Y(t-1)) \text{ C } (Y(t) \text{ B } \overline{Y}(t-1))$, где А, В, С - соответствующие знаки булевых функций.

9. Двухзначные двоичные числа $c = a_{13}a_{14}$, $d = a_{15}a_{16}$, $g = a_{17}a_{18}$, $h = a_{19}a_{20}$ перевести в десятичную систему.
 Вычислить $m = c + 3$, $n = c + d + 4$, $p = g + 3$, $q = g + h + 3$.
 Заполнить таблицу К8.

Таблица К8

c	d	g	h	m	n	p	q

Найти число следующих комбинаторных конфигураций:

$$A_n^m, C_n^m, A_n^m, C_n^m, A_m^n, C_m^n,$$

$$A_q^p, C_q^p, A_q^p, C_q^p, A_p^q, C_p^q.$$

ТРЕНИНГ УМЕНИЙ

1. Пример выполнения упражнения тренинга на умение №1.

Задание

Ориентированный граф G с множеством вершин $V = \{1, 2, 3, 4, 5, 6, 7\}$ задан списком дуг $E = \{(1, 4) (1, 6) (2, 1) (2, 2) (2, 6) (3, 2) (3, 4) (4, 6) (5, 2) (5, 4) (5, 4) (5, 5) (6, 2) (6, 5) (7, 1) (7, 6)\}$.

Построить реализацию графа G .

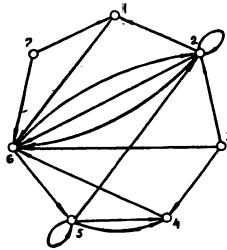
Построить матрицу инцидентий графа G .

Построить матрицу соседства вершин графа G .

Для соотнесенного неориентированного графа G построить матрицу соседства вершин.

Решение

Заполните таблицу, подобрав алгоритму конкретное содержание.

№ п/п	Алгоритм	Конкретное соответствие задания заданному алгоритму
1	Построить реализацию графа G	Изображения вершин 1, 2, ..., 7 удобно поместить (приблизительно) в вершинах правильного 7-угольника. Соединить пары вершин в соответствии со списком дуг, избегая пересечения трех ребер в одной точке. Обозначить стрелками ориентацию дуг. 
2	Построить матрицу инцидентий графа G	Число ребер графа $G - 18$. В прямоугольной матрице $A = \ a_{ij}\$ размерности 7×18 последовательно заполнять столбцы в соответствии со списком дуг: каждой дуге $e_k = (i, j)$, где $i \neq j$ соответствуют элементы $a_{ik} = -1$ и $a_{jk} = +1$. Петле $e_k = (i, i)$ соответствует элемент $a_{ik} = 2$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	-1	-1	1														1	
2			-1	2	-1	-1	1				1				1			
3							-1	-1	-1									
4	1							1		-1		1	1					
5											-1	-1	-1	2		1		
6		1			1	1			1	1					-1	-1		1
7																	-1	-1

3	Построить матрицу соседства вершин графа G	В квадратной матрице $B = \ b_{ij}\ $ размерности 7×7 каждой дуге $e_k = (i, j)$ соответствует элемент $b_{ij} = 1$, кроме кратных дуг, которым соответствует элемент, равный кратности дуги (i, j) .
---	--	---

	1	2	3	4	5	6	7
1	0	0	0	1	0	1	0
2	1	1	0	0	0	2	0
3	0	1	0	1	0	1	0
4	0	0	0	0	0	1	0
5	0	1	0	2	1	0	0
6	0	1	0	0	1	0	0
7	1	0	0	0	0	1	0

4	Для соотнесенного неориентированного графа G построить матрицу соседства вершин	Квадратная матрица $C = \ c_{ij}\ $ размерности 7×7 строится из предыдущей матрицы B следующим образом: $c_{ij} = c_{ji} = b_{ij} + b_{ji}$ для $i \neq j$; $c_{ii} = b_{ii}$.
---	---	---

	1	2	3	4	5	6	7
1	0	1	0	1	0	1	1
2	1	1	1	0	1	3	0
3	0	1	0	1	0	1	0
4	1	0	1	0	2	1	0
5	0	1	0	2	1	1	0
6	1	3	1	1	1	0	1
7	1	0	0	0	0	1	0

Постройте самостоятельно различные представления графов, заданных списком дуг:

1. $\{(1, 6), (2, 1), (2, 3), (3, 1), (3, 3), (3, 3), (3, 4), (3, 6), (5, 1), (5, 6), (5, 6), (5, 6), (7, 4), (7, 6)\}$;
2. $\{(1, 2), (1, 4), (1, 5), (2, 4), (3, 2), (3, 4), (3, 4), (4, 2), (4, 5), (5, 5), (5, 7), (7, 1)\}$;

3. $\{(1, 2), (1, 6), (2, 2), (2, 3), (3, 1), (3, 2), (3, 6), (4, 3), (4, 3), (4, 5), (4, 6), (5, 1), (5, 2), (5, 4), (5, 6)\}$;
 4. $\{(1, 4), (2, 1), (2, 5), (2, 6), (3, 4), (3, 7), (4, 1), (5, 3), (5, 4), (5, 6), (7, 1), (7, 5)\}$;
 5. $\{(2, 5), (2, 1), (2, 6), (2, 7), (3, 5), (3, 5), (4, 1), (4, 3), (4, 5), (6, 1), (6, 2), (6, 5), (6, 7), (7, 3)\}$.

2. Пример выполнения упражнения тренинга на умение №2.

Задание

Неориентированный граф G с множеством вершин $V = \{1, 2, 3, 4, 5, 6, 7\}$ задан списком ребер $E = \{(1, 2), (1, 4), (1, 6), (1, 7), (2, 3), (2, 5), (2, 6), (3, 4), (3, 6), (4, 5), (4, 6), (5, 6), (5, 7)\}$.

Построить реализацию графа G .

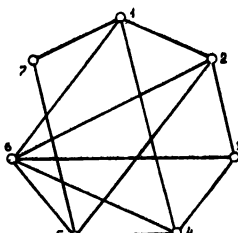
Найти цикломатическое число графа G .

Выбрать остов графа G .

Построить базис циклов графа G .

Решение

Заполните таблицу, подобрав алгоритму конкретное содержание.

№ п/п	Алгоритм	Конкретное соответствие задания заданному алгоритму
1	Построить реализацию графа G	Процесс построения реализации графа описан в тренинге умения 1. 
2	Найти цикломатическое число графа G	Цикломатическое число, равное размерности базиса циклов графа, определяется формулой $\nu = p - b + k$, где p – число ребер, b – число вершин, k – число компонент связности графа. $p = 13$, $b = 7$, $k = 1$ (нетрудно убедиться, что граф G – связный). Следовательно, $\nu = 13 - 7 + 1 = 7$.
3	Выбрать остов графа G	Поскольку число вершин графа G равно 7, в любом его остове должно быть 6 ребер. При заданном упорядочении вершин 1 – 7, начиная с вершины 1, последовательно добавляем ребра, присоединяющие к строящемуся дереву новую вершину: (1, 2), (1, 4), (1, 6), (1, 7), (2, 3), (2, 5).

4 Построить базис циклов графа G

1. Составим список хорд относительно построенного остова (все остальные ребра графа): (2, 6), (3, 4), (3, 6), (4, 5), (4, 6), (5, 6), (6, 7).

Число хорд должно быть равно цикломатическому числу графа, т.е. 7.

2. Поочередно присоединяя к остову по одной хорде, выделяем элементарный цикл, состоящий из этой хорды и части ребер остова – единственной цепи, связывающей в остове концы хорды.

Хорда	Добавляемая цепь
* (2, 6)	(6, 1), (1, 2)
(3, 4)	(4, 1), (1, 2), (2, 3)
(3, 6)	(6, 1), (2, 1), (2, 3)
* (4, 5)	(5, 2), (2, 1), (1, 4)
(4, 6)	(6, 1), (1, 4)
* (5, 6)	(6, 1), (1, 2), (2, 5)
(6, 7)	(7, 1), (1, 6)

3. Выразим в построенном базисе элементарный цикл $C = (6, 2, 1, 4, 5, 6)$. Он получается сложением по модулю 2 тех базисных циклов, которые содержат хорды, принадлежащие C (они отмечены звездочкой в предыдущей таблице: (6, 2), (4, 5), (5, 6)).

Проверьте, что ребра цикла C содержатся в совокупности базисных циклов нечетное число раз (в частности, все хорды – ровно по одному разу), а ребра, не принадлежащие C (например, ребро (5, 2)), – четное.

Постройте самостоятельно остов и базис циклов неориентированных графов, заданных списком ребер:

1. $\{(1, 2), (1, 3), (1, 5), (1, 6), (2, 3), (2, 4), (3, 4), (3, 6), (4, 7), (5, 6), (6, 7)\};$
2. $\{(1, 2), (1, 4), (1, 5), (1, 7), (2, 3), (2, 4), (3, 4), (3, 5), (3, 7), (4, 5), (5, 7)\};$
3. $\{(1, 2), (1, 3), (1, 5), (1, 6), (2, 3), (2, 5), (3, 4), (3, 6), (4, 5), (4, 6), (5, 6)\};$
4. $\{(1, 2), (1, 4), (1, 5), (1, 7), (2, 5), (2, 6), (3, 4), (3, 5), (3, 7), (4, 5), (5, 6), (5, 7)\};$
5. $\{(1, 2), (1, 4), (1, 6), (1, 7), (2, 5), (2, 6), (2, 7), (3, 4), (3, 5), (3, 7), (4, 5), (5, 6), (6, 7)\}.$

Определите цикломатическое число графа.

3. Пример выполнения упражнения тренинга на умение №3.

Задание

Для списка сообщений с заданным распределением частот построить код Фано. Определить стоимость кода.

S	T	U	V	W	X	Y	Z
0,08	0,1	0,15	0,15	0,3	0,05	0,12	0,05

Решение

Заполните таблицу, подобрав алгоритму конкретное содержание.

№ п/п	Алгоритм	Конкретное соответствие задания заданному алгоритму							
1	Упорядочить список сообщений по убыванию частот. При равенстве частот порядок может быть любым.	W	U	V	Y	T	S	X	Z
		0,3	0,15	0,15	0,12	0,1	0,08	0,05	0,05
2	Разделить группу событий на 2 части, различающиеся возможно меньше по суммарной частоте.	W	U	V	Y	T	S	X	Z
		0,3	0,15	0,15	0,12	0,1	0,08	0,05	0,05
		В нашем примере разделение (W, U) – (V, ..., Z) дает соотношение частот 0,45 : 0,55 с разницей 0,1. Другое разделение (W, U, V) – (Y, ..., Z) дает соотношение частот 0,6 : 0,4 с большей разницей (0,2).							
3	Установить первый символ кода 0 для сообщений первой группы, 1 – для второй группы.	W 0,3	0	U 0,15	0	V 0,15	1	Y 0,12	1
		T 0,1	1	S 0,08	1	X 0,05	1	Z 0,05	1

4	Повторять процедуру п.3 деления на две части в каждой из групп до тех пор, пока в группе больше одной строки, добавляя каждый раз к коду справа 0 для первой части подразделения и 1 для второй части.	W 0.3 } 0.45 U 0.15 } 0.15 V 0.15 } 0.27 Y 0.12 } 0.12 T 0.1 } 0.1 S 0.08 } 0.55 X 0.05 } 0.28 Z 0.05 } 0.18 0.3 0.15 0.15 0.12 0.1 0.08 0.1 0.05 00 01 100 101 110 1110 11110 11111 Для 4 последних строк выбрано разделение T – (S, X, Z) с соотношением частот 0,1 : 0,18 (0,08 + 0,05 + 0,05); возможно и другое разделение (T, S) – (X, Z) с соотношением частот 0,18 : 0,1 ((0,1 + 0,08) : (0,05 + 0,05)), что даст другой вариант кодирования этих сообщений: T 0,1 S 0,08 X 0,05 Z 0,05 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1
5	Определить стоимость построения кода	Для первого варианта кода стоимость равна $2 \cdot (0,3 + 0,15) + 3 \cdot (0,15 + 0,12 + 0,1) + 4 \cdot 0,08 + 5 \cdot (0,05 + 0,05) = 2,83.$ Для второго варианта стоимость равна $2 \cdot (0,3 + 0,15) + 3 \cdot (0,15 + 0,12) + 4 \cdot (0,1 + 0,08 + 0,05 + 0,05) = 2,83$ В этом примере оба варианта имеют одинаковую стоимость.

Постройте самостоятельно код Фано для списка сообщений с заданным распределением частот. Определите стоимость кода.

	S	T	U	V	W	X	Y	Z
1.	0,15	0,1	0,15	0,15	0,1	0,1	0,15	0,1
2.	0,15	0,02	0,25	0,15	0,08	0,15	0,2	-
3.	0,02	0,25	0,04	0,01	0,4	0,1	0,03	0,15
4.	0,15	0,2	0,08	0,12	0,15	0,1	0,2	-
5.	0,07	0,04	0,3	0,1	0,07	0,12	0,3	-

4. Пример выполнения упражнения тренинга на умение №4.

Задание

Для заданного сообщения $X = 0\ 1\ 1\ 0\ 1\ 0\ 1$ построить код Хэмминга X' . Внести одиночную ошибку замещения и произвести декодирование.

Решение

Заполните таблицу, подобрав алгоритму конкретное содержание.

№ п/п	Алгоритм	Конкретное соответствие задания заданному алгоритму
1	Подготовить строку для сообщения X' , отводя места с номерами, равными 2^k , для проверочных символов, а остальные разряды – для информационных символов сообщения X' .	Разряды 1, 2, 4, 8 – проверочные; разряды 3, 5, 6, 7, 9, 10, 11 ... - информационные 1 2 3 4 5 6 7 8 9 10 11 п п п и и и п и и
2	Разместить знаки сообщения X в информационных разрядах.	1 2 3 4 5 6 7 8 9 10 11 ----- 0 1 1 0 1 0 1

3	Построить таблицу с двоичными номерами разрядов и внести в информационные разряды знаки сообщения X.	<table><tr><td>1</td><td>0001</td><td></td></tr><tr><td>2</td><td>0010</td><td></td></tr><tr><td>3</td><td>0011</td><td>0</td></tr><tr><td>4</td><td>0100</td><td></td></tr><tr><td>5</td><td>0101</td><td>1</td></tr><tr><td>6</td><td>0110</td><td>1</td></tr><tr><td>7</td><td>0111</td><td>0</td></tr><tr><td>8</td><td>1000</td><td></td></tr><tr><td>9</td><td>1001</td><td>1</td></tr><tr><td>10</td><td>1010</td><td>0</td></tr><tr><td>11</td><td>1011</td><td>1</td></tr></table>	1	0001		2	0010		3	0011	0	4	0100		5	0101	1	6	0110	1	7	0111	0	8	1000		9	1001	1	10	1010	0	11	1011	1																											
1	0001																																																													
2	0010																																																													
3	0011	0																																																												
4	0100																																																													
5	0101	1																																																												
6	0110	1																																																												
7	0111	0																																																												
8	1000																																																													
9	1001	1																																																												
10	1010	0																																																												
11	1011	1																																																												
4	Для каждого из проверочных разрядов с номером 2^k определить строки, формирующие проверочные знаки кода X': 1) информационный знак кода X' равен 1; 2) в двоичном представлении номера строки k-й разряд равен 1.	<table><tr><th>№ строки</th><th>Двоичное представление номера строки</th><th>Инф. знаки</th><th>Провер. знаки</th><th>Номера строк, формирующих провер. знак</th></tr><tr><td>1</td><td>0001</td><td></td><td>1</td><td>5 + 9 + 11</td></tr><tr><td>2</td><td>0010</td><td></td><td>0</td><td>6 + 11</td></tr><tr><td>3</td><td>0011</td><td>0</td><td></td><td></td></tr><tr><td>4</td><td>0100</td><td></td><td>0</td><td>5 + 6</td></tr><tr><td>5</td><td>0101</td><td>1</td><td></td><td></td></tr><tr><td>6</td><td>0110</td><td>1</td><td></td><td></td></tr><tr><td>7</td><td>0111</td><td>0</td><td></td><td></td></tr><tr><td>8</td><td>1000</td><td></td><td>0</td><td>9 + 11</td></tr><tr><td>9</td><td>1001</td><td>1</td><td></td><td></td></tr><tr><td>10</td><td>1010</td><td>0</td><td></td><td></td></tr><tr><td>11</td><td>1011</td><td>1</td><td></td><td></td></tr></table> В таблице выделены жирным шрифтом единицы в тех строках, в которых значение исходного сообщения равно 1 (строки 5, 6, 9, 11).	№ строки	Двоичное представление номера строки	Инф. знаки	Провер. знаки	Номера строк, формирующих провер. знак	1	0001		1	5 + 9 + 11	2	0010		0	6 + 11	3	0011	0			4	0100		0	5 + 6	5	0101	1			6	0110	1			7	0111	0			8	1000		0	9 + 11	9	1001	1			10	1010	0			11	1011	1		
№ строки	Двоичное представление номера строки	Инф. знаки	Провер. знаки	Номера строк, формирующих провер. знак																																																										
1	0001		1	5 + 9 + 11																																																										
2	0010		0	6 + 11																																																										
3	0011	0																																																												
4	0100		0	5 + 6																																																										
5	0101	1																																																												
6	0110	1																																																												
7	0111	0																																																												
8	1000		0	9 + 11																																																										
9	1001	1																																																												
10	1010	0																																																												
11	1011	1																																																												
5	Вычислить значения проверочных разрядов.	Значения проверочных разрядов получаются сложением по модулю 2 значений соответствующих информационных разрядов. Значение проверочного разряда определяется четностью числа слагаемых. Результат расчета – в предпоследнем столбце таблицы.																																																												
6	Объединяя информационные и проверочные разряды, получить искоемое сообщение X'.	Построенное закодированное по Хэммингу сообщение $X' = 10001100101$. Проверим правильность кодирования, вычислив значение $H(X)$. Выпишем в столбец двоичные номера строк, в которых знак сообщения X' равен единице. Сумма по модулю 2 знаков номеров в каждом разряде должна быть равной 0.																																																												

		№ строки	Двоичное представление номера строки	Знаки со- общ. X'
		1	0 0 0 1	1
		5	0 1 0 1	1
		6	0 1 1 0	1
		9	1 0 0 1	1
		11	1 0 1 1	1
			0 0 0 0	
7	Внести ошибку замещения в один из разрядов и произвести декодирование.	Пусть при передаче сообщения X' произошла ошибка замещения в 7-м разряде, т.е. полученное сообщение X'' - 1 0 0 0 1 1 1 0 1 0 1. Вычислим значение $H(X'')$. Выпишем в столбец двоичные номера строк, в которых знак сообщения X'' равен единице. Суммируем по модулю 2 знаки номеров в каждом разряде.		
		<pre> 0 0 0 1 1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 0 1 1 1 ----- 0 1 1 1 </pre> Полученное двоичное число 0111 равно номеру разряда 7, в котором произошла ошибка. Заменяя в сообщении X'' значение 7-го разряда на противоположное, восстанавливаем X'; вычеркивая из X' проверочные разряды, получаем искомое сообщение X.		

Постройте самостоятельно код Хэмминга X' для заданного сообщения X . Внесите ошибку в i -й разряд и, произведя декодирование, подтвердите место ошибки:

1. $X = 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0$ ($i = 6$)
2. $X = 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1$ ($i = 4$)
3. $X = 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1$ ($i = 9$)
4. $X = 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1$ ($i = 10$)
5. $X = 1\ 0\ 1\ 0\ 0\ 1\ 1$ ($i = 5$)

5. Пример выполнения упражнения тренинга на умение №5.

Задание

Построить автомат, реализующий оператор, заданный уравнением, выражающим зависимость выходного значения от входной последовательности в алфавите $\{0, 1\}$:

$$Z(t) = \begin{cases} 1, \text{ если } X(t) = \overline{X(t-1)} = X(t-2) = 1 \\ 0 \text{ в противном случае} \end{cases}$$

Решение

Заполните таблицу, подобрав алгоритму конкретное содержание.

№ п/п	Алгоритм	Конкретное соответствие задания заданному алгоритму
1	Определить входной и выходной алфавиты.	Входной и выходной алфавиты определяются заданием оператора $Z(t)$: двухбуквенный алфавит $\{0,1\}$.
2	Определить необходимое множество Q внутренних состояний автомата, представляющих различные ситуации, требующие различной реакции. Для каждого состояния определить выходные значения и назначить переходы в следующее состояние в зависимости от входного значения.	$Z(t) = 1$, если последние 3 входных символа образуют подслово 101. Для определения $Z(t)$ требуется запоминание двух предшествующих входных значений. Можно обозначить через q_1 состояние, фиксирующее первую (возможную) единицу $X(\tau) = 1$. При $X(\tau+1) = 0$ автомат переходит в состояние, которое можно обозначить q_{10} , а при $X(\tau+1) = 1$ предыдущая единица уже не может быть первой единицей фрагмента 101, но возможно таковой является текущая единица, поэтому автомат остается в состоянии q_1 . Из состояния q_{10} при $X(\tau+2) = 0$ автомат переходит в состояние q_0 , предшествующее возможному фрагменту 101. Если же $X(\tau+2) = 1$, то, во-первых, $Z=1$ (при всех остальных переходах во всех состояниях $Z = 0$), а во-вторых, это значение $X(\tau+2)$ может восприниматься как потенциальная первая единица следующего фрагмента 101; поэтому переход – в состояние q_1 . Таким образом, $Q = \{q_0, q_1, q_{10}\}$.

3	Построить таблицу размерности $w \times p$ (w – число входных символов; p – число внутренних состояний автомата)	Таблица имеет размерность 3×2		
4	Построить таблицу (матрицу) переходов автомата, занося в клетки таблицы определенные входным символом и состоянием автомата символ следующего состояния и выходное значение.			
			0	1
		q_0	$q_0 / 0$	$q_1 / 0$
		q_1	$q_{10} / 0$	$q_1 / 0$
		q_{10}	$q_0 / 0$	$q_1 / 1$
5	Построить граф (диаграмму) переходов автомата.	Преобразуем построенную таблицу в граф переходов с множеством вершин Q и дугами, соответствующими входному символу и направленными в следующее состояние, с указанием входного значения.		

Решите самостоятельно следующие задачи.

1. Постройте графы переходов для автоматов, реализующих следующие операторы:

$$а) Z(t) = \begin{cases} \overline{X}(t), & \text{если } t \leq 2, \\ X(t - 1), & \text{если } t > 2. \end{cases}$$

Указание: для $t = 0, 1$ достаточно по одному состоянию, для $t = 2, 3, \dots$ нужны 2 состояния, учитывающие значения $X(t), X(t - 1)$.

б) $Z(t)$ реализуется логической сетью, изображенной на рис. 4.12 (гл. IV, § 2, п. 1, пример 4).

$$в) Z(t) = X(t) Y(t - 1) \vee X(t - 1) Y(t).$$

Указание: построить логическую сеть и представить ее как автомат (см. гл IV, § 2, п. 4).

2. Постройте таблицы переходов для этих автоматов.

6. Пример выполнения упражнения тренинга на умение №6.

Задание

Построить таблицу переходов и граф переходов для машины Тьюринга, заданной программой с внешним алфавитом $\{a, b, c, 0\}$ и внутренним алфавитом (множеством состояний) $\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$: $q_1a \rightarrow q_2R$, $q_1b \rightarrow R$, $q_10 \rightarrow q_6L$, $q_2a \rightarrow q_3cR$, $q_2b \rightarrow q_4cR$, $q_3a \rightarrow aR$, $q_3b \rightarrow q_4aR$, $q_30 \rightarrow q_5aL$, $q_4a \rightarrow q_3bR$, $q_4b \rightarrow bR$, $q_40 \rightarrow q_5bL$, $q_5a \rightarrow L$, $q_5b \rightarrow L$, $q_5c \rightarrow q_1aR$, $q_6a \rightarrow L$, $q_6b \rightarrow L$, $q_6c \rightarrow aL$, $q_60 \rightarrow q_0R$.

Применить МТ к заданной начальной конфигурации $q_1abbaab$: составить протокол применения, определить заключительную конфигурацию.

Решение

Заполните таблицу, подобрав алгоритму конкретное содержание.

№ п/п	Алгоритм	Конкретное соответствие задания заданному алгоритму																																			
1	Построить таблицу переходов МТ.	Программа МТ записана в сокращенной форме: в правой части команд опущены символы, не изменяющиеся при выполнении данной команды, т.е. совпадающие с соответствующими символами левой части. Напомним, что 0 обозначает пустой символ. Каждую команду программы переносим в таблицу. <table><tr><th></th><th>a</th><th>b</th><th>c</th><th>0</th></tr><tr><td>q₁</td><td>q₂ R</td><td>R</td><td></td><td>q₆ L</td></tr><tr><td>q₂</td><td>q₃ c R</td><td>q₄ c R</td><td></td><td></td></tr><tr><td>q₃</td><td>a R</td><td>q₄ a R</td><td></td><td>q₅ a L</td></tr><tr><td>q₄</td><td>q₃ b R</td><td>b R</td><td></td><td>q₅ b L</td></tr><tr><td>q₅</td><td>L</td><td>L</td><td>q₁ a R</td><td></td></tr><tr><td>q₆</td><td>L</td><td>L</td><td>a L</td><td>q₀ R</td></tr></table> Некоторые клетки таблицы не заполнены, поскольку соответствующие им ситуации не возникают при применении МТ к данной начальной конфигурации.		a	b	c	0	q ₁	q ₂ R	R		q ₆ L	q ₂	q ₃ c R	q ₄ c R			q ₃	a R	q ₄ a R		q ₅ a L	q ₄	q ₃ b R	b R		q ₅ b L	q ₅	L	L	q ₁ a R		q ₆	L	L	a L	q ₀ R
	a	b	c	0																																	
q ₁	q ₂ R	R		q ₆ L																																	
q ₂	q ₃ c R	q ₄ c R																																			
q ₃	a R	q ₄ a R		q ₅ a L																																	
q ₄	q ₃ b R	b R		q ₅ b L																																	
q ₅	L	L	q ₁ a R																																		
q ₆	L	L	a L	q ₀ R																																	

2	Построить граф переходов МТ.	Граф переходов содержит 7 вершин: $q_0, q_1, q_2, q_3, q_4, q_5, q_6$. Из каждой вершины исходит столько дуг, сколько команд в соответствующей строке таблицы, однако некоторые кратные дуги могут быть склеены, если у них одинаковая правая часть: например, для пары ситуаций q_5a, q_5b, а также для пары q_6a, q_6b.
3	Составить протокол применения МТ к начальной конфигурации	Протокол представляет собой последовательность $K_1 \rightarrow K_2 \rightarrow K_3 \dots$ конфигураций (слово на ленте и положение головки МТ), где K_1 – начальная конфигурация, а каждая из конфигураций K_2, K_3 и т.д. получается из предыдущей в результате применения программы м/т. 5 первых конфигураций для данного примера таковы: $q_1 a b b a a b$ $a q_2 b b a a b$ $a c q_4 b a a b$ $a c b q_4 a a b$ $a c b b q_3 a b$ Для компактности протокола и удобства рассмотрения будем изображать конфигурацию словом во внешнем алфавите, а положение головки - подстрочным индексом, т.е. символ x_i обозначает головку в состоянии q_i на ячейке ленты с символом x: вместо фрагмента $\dots r q_i s t \dots$ будем писать $\dots r s_i t \dots$. Кроме того, поскольку каждый шаг работы МТ изменяет только ячейку, в которой располагается головка, можно не повторять в каждой строке протокола всю конфигурацию, а приводить лишь символы в окрестности головки, выделяя жирным шрифтом изменяющийся символ. Целесообразно время от времени изображать полную конфигурацию.

Отметим также, что слева и справа от слова на ленте располагаются пустые символы, которые обозначаются нулем в том случае, если в ячейке с пустым символом находится головка МТ. Протокол в этой форме выглядит так (несколько первых строк приведены в полной форме):

```

a1 b b a a b
a b2 b a a b
a c b4 a a b
a c b a4 a b
      b b a3 b
            a b3
            a 04
            a5 b
            a5 a
            b5 a
            b5 b
a 05 b b a a b
a a b1 b
      b b1 a
      b a1 a
            a a2 b
            a c b3
a a b b a c a 04
            c a5 b
            a 05 a
            a a a1 b
            a b2
            c 04
            a 05 b
            a b1
a a b b a a a a b 01
            a b6
            a a6 b
            a a6 a
            a a6 a
            b a6 a
            b b6 a
            a b6 b
            a a6 b
            a6 a
06 a
a0 a b b a a a a b

```

Протокол заканчивается по достижении заключительного состояния q_0 . Последняя строка протокола изображает заключительную конфигурацию, которую можно переписать в стандартной форме:

$q_0 a a b b a a a b$

Решите самостоятельно следующие задачи.

1. Составить протоколы применения данной МТ к следующим начальным конфигурациям:

- а) $q_1 b a b a$
- б) $q_1 b a a a b$
- в) $q_1 a a b a b b$
- г) $q_1 a b a a b a$
- д) $q_1 a b b b a a b$

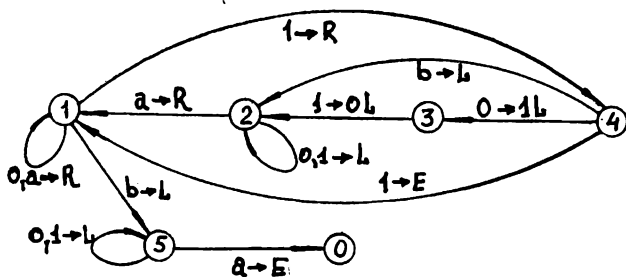
2. Построить граф переходов для МТ, заданной таблицей:

	0	1	a	b
q_1	$q_3 R$	R	R	$q_5 R$
q_2	$q_4 1 L$			
q_3	q_1	$q_2 0 L$		$q_4 R$
q_4	L	L	$q_1 R$	
q_5	L	L	q_0	

Составить протоколы применения данной МТ к следующим начальным конфигурациям:

- а) $q_1 a 0 1 0 0 1 b$;
- б) $q_1 a 1 0 0 1 0 b$;
- в) $q_1 a 0 0 1 1 1 b$.

3. Построить таблицу переходов для МТ, заданной графом:



Составить протоколы применения данной МТ к следующим начальным конфигурациям:

а) $q_1 a 1 0 0 1 0 b$;

б) $q_1 a 0 1 0 0 1 b$;

в) $q_1 a 1 1 0 0 0 b$.

ДИСКРЕТНАЯ МАТЕМАТИКА

ЮНИТА 2

ГРАФЫ И СЕТИ. КОДИРОВАНИЕ. АВТОМАТЫ И АЛГОРИТМЫ

Редактор Н.И. Лебедева

Оператор компьютерной верстки А.Б. Кондратьева

Изд. лиц. ЛР № 071765 от 07.12.1998

Сдано в печать 26.11.2001 г.

НОУ "Современный Гуманитарный Институт"

Уч.-изд. л. 9,56

Тираж 7000. Заказ 4103.

Современный Гуманитарный Университет

Отпечатано в Коломенской типографии. 140400, Московская обл., г. Коломна, ул. III Интернационала, д. 2 а.

1 ~~B~~ 10 ~~B~~ v +

2 ~~A~~ 11 ~~A~~ v +

3 C 12 C

4 C 13 C

5 ~~D~~ 14 ~~B~~

6 B 15 A

16 B

7 C 17 C v

8 ~~D~~ 18 ~~B~~

9 B

19 ~~D~~ v +

20 C v

21 ~~A~~ v +

22 ~~D~~ v +

23 A

24 ~~A~~ v +

25 B v.